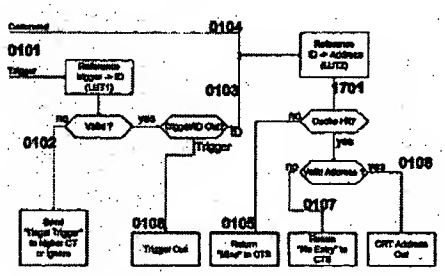
Method of managing configuration data in data flow processors

Publication number: DE19807872
Publication date: 1999-08-26
Inventor: VORBACH MARTIN (DE); MUENCH ROBERT (DE)
Applicant: PACT INF TECH GMBH (DE)
Classification:
- international: *G06F12/08; G06F15/78; G06F15/82; H03K19/173;* G06F12/08; G06F15/76; H03K19/173; (IPC1-7): G06F15/80; G06F9/445; G11C16/02
- European: G06F15/78R
Application number: DE19981007872 19980225
Priority number(s): DE19981007872 19980225

Also published as:
WO9944147 (A3)
WO9944147 (A2)
WO9944120 (A3)
WO9944120 (A2)
EP1057117 (A3)
more >>

Abstract of **DE19807872**

The method involves one unit managing an associated number equal to some or all of a total number of configurable elements. Reconfiguration requests are sent from the elements to the unit. The unit accepts no further requests and changes during processing; loads yet to be loaded configuration data of earlier requests from an intermediate memory into the configurable element; and converts the current request to a definite identifier or ID. The ID is converted to the address in the unit's memory of the configuration data to be loaded if the data exists in the memory. The data are requested from a superior unit if not present in the unit's memory. The data are loaded if the element can accept them. Data which cannot be accepted are placed in temporary memory. When configuration data have been fully processed, new requests can be accepted, until when the configuration data yet to be loaded from earlier request are loaded into the configurable elements.



(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) **Offenlegungsschrift**

(10) **DE 198 07 872 A 1**

(51) Int. Cl.[6]:
G 06 F 15/80
G 11 C 16/02
G 06 F 9/445

(21) Aktenzeichen: 198 07 872.2
(22) Anmeldetag: 25. 2. 98
(43) Offenlegungstag: 26. 8. 99

(71) Anmelder:
Pact Informationstechnologie GmbH, 81545 München, DE

(74) Vertreter:
Pietruk, C., Dipl.-Phys., Pat.-Anw., 76229 Karlsruhe

(72) Erfinder:
Vorbach, Martin, 76149 Karlsruhe, DE; Münch, Robert, 76149 Karlsruhe, DE

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Verfahren zur Verwaltung von Konfigurationsdaten in Datenflußprozessoren sowie Bausteinen mit zwei- oder mehrdimensionalen programmierbaren Zellstruktur (FPGAs, DPGAs, o. dgl.

DE 198 07 872 A 1

DE 198 07 872 A 1

Beschreibung

Hintergrund der Erfindung

Stand der Technik

Der Stand der Technik, welcher diese Patentschrift zugrundeliegt, ist durch die Patentanmeldung 196 54 846.2-53 (Verfahren zum selbständigen dynamischen Umladen von Datenflußprozessoren (DFPs) sowie Bausteinen mit zwei- oder mehrdimensionaler programmierbaren Zellmatrix (FPGAs, DPGAs, o. dgl.) und der Patentanmeldung 196 54 593.5–53 (Umkonfigurierungs Verfahren für programmierbare Bausteine zur Laufzeit) beschrieben. Darin wird ein Verfahren zur Konfiguration und Umkonfiguration von DFPs, sowie FPGAs, DPGAs und ähnlichen Bausteinen nach dem Stand der Technik, beschrieben, bei dem ein separat ausgestalteter zentral übergeordneter Mikrokontroller-ähnlicher Baustein die Verteilung von Konfigurationsdaten an mehrere untergeordnete, weitgehend passive Steuereinheiten übernimmt.

Probleme

Durch den Einsatz einer zentralen und globalen Einheit, welche die Rekonfiguration von Teilen eines oder mehrerer Bausteine steuert, kommt es zu Engpässen, wenn viele verschiedene Rekonfigurations-Anfragen auf einmal behandelt werden müssen. Die Vorteile der Parallelität, der beschriebenen Bausteine, wird durch eine solche zentrale Einheit stark eingeschränkt, da er den typischen "Flaschenhals" darstellt und die Verarbeitung der Daten dadurch erheblich verlangsamt. Weiterhin ist die Zuordnung der Ereignisquelle zu der zu ladenden Konfiguration problematisch, da mit absoluten Adressen des Konfigurationsspeichers gearbeitet wird. Die Rekonfigurations-Einheit muß also eine Art Speicherverwaltungssystem beinhalten, welche, ähnlich wie in einem Betriebssystem, mitprotokolliert, welche Speicherbereich von welcher Konfiguration benutzt werden.

Verbesserung durch die Erfindung, Aufgabe

Die Grundaufgabe der vorliegenden Erfindung ist eine Einheit – im Folgenden Konfigurationstabelle (CT) genannt –, die hierarchisch aufgebaut ist und auf jeder Ebene mehrfach vorkommen kann, wobei sich die Zahl der CTs von der untersten Hierarchiestufe zur obersten so verringert, daß auf der höchsten Ebene genau eine CT vorhanden ist. Jede CT konfiguriert und kontrolliert unabhängig von anderen und parallel eine Mehrzahl von konfigurierbaren Elementen (CELs). CTs höherer Hierarchiestufen können Konfigurationsroutinen für tieferliegende CTs zwischenspeichern. Benötigen mehrere der tieferliegenden CTs ein und dieselbe Konfigurationsroutine, wird diese bei einer höherliegenden CT zwischengespeichert und von den einzelnen CTs abgerufen, wobei die höherliegende CT die betreffende Konfigurationsroutine nur ein Mal aus einem globalen gemeinsamen Konfigurationsspeicher abruft, wodurch ein Cache-Effekt erzielt wird. Abseits konfigurierbarer Bausteine kann die vorliegende Erfindung als Cacheverfahren für Instruktions- und Datencache in Mikroprozessoren, DFP oder dgl. mit mehreren Rechenwerken eingesetzt werden. Dabei können, je nach Anwendung, einige der im Folgenden beschriebenen Einheiten entfallen (z. B. FILMO), am hierarchischen Aufbau ändert sich jedoch grundlegend nichts. Deshalb wird dieser Einsatz als eine Teilmenge betrachtet und nicht weiter darauf eingegangen.

Beschreibung der Erfindung

Anstatt wie bisher eine zentrale und globale Einheit in einen Baustein zu integrieren, welche alle Konfigurations-Anforderungen bearbeitet, existieren nun eine Mehrzahl von hierarchisch (Baum-/Waldstruktur) angeordneten aktiven Einheiten, welche diese Aufgabe übernehmen können. Dabei wird eine Anforderung von der tiefesten Ebene (den Blättern in der Hierarchie) nur dann an die nächst höher gelegene Ebene weitergeleitet, wenn die Anforderung nicht bearbeitet werden konnte. Diese Schritte werden für alle vorhandenen Ebenen wiederholt, bis die höchst gelegene Ebene erreicht ist.

Die höchst gelegene Ebene ist an einen internen oder externen übergeordneten Konfigurationspeicher angeschlossen, der alle jemals für diesen Programlauf, benötigten Konfigurationsdaten enthält.

Durch die Baumstruktur der Konfigurationseinheiten wird eine Art Cacheing der Konfigurationsdaten erreicht. Zugriffe auf Konfigurationen finden hauptsächlich lokal statt. Im ungünstigsten Fall muß eine Konfiguration aus den übergeordneten Konfigurationsspeicher geladen werden, falls die betreffenden Daten in keiner der hierarchisch angeordneten CTs vorhanden sind.

Die Grundlagen der CT

Eine Konfigurationstabelle (CT) ist eine aktive Einheit, die auf Synchronisationssignale, sogenannten Trigger (siehe DE 197 04 728.9), reagiert. Die Trigger werden von einer zwei- oder mehrdimensionalen Matrix aus elektronischen Baugruppen, für gewöhnlich arithmentischen oder logischen Einheiten, Adressgeneratoren (siehe DE 196 54 595.1–53), Recheneinheiten (siehe DE 196 51 075.9), o.a. – im Folgenden konfigurierbare Elemente (CEL) genannt – generiert. Anhand des auftretenden Trigger wird eine bestimmte Aktion innerhalb der CT ausgelöst. Dabei ist es Aufgabe der CT die Steuerung einer Mehrzahl von CELs zu übernehmen und deren arithmetischen und/oder logischen Operationen zu bestimmen. Insbesondere müssen CELs, entsprechend DE 196 51 075.9 konfiguriert und umkonfiguriert werden. Diese Aufgabe übernimmt eine CT, indem sie eine Mehrzahl von möglichen Konfigurationsroutinen (KK), die ihrerseits jeweils aus einer Mehrzahl von einzelnen Konfigurationsworten (KW) bestehen, verwaltet und eine Mehrzahl von CELs

aufgrund von Triggerbedingungen mit einer oder mehrerer der KP konfigurieren. Dabei erhält eine CEL jeweils eines oder mehrere der Konfigurationsworte, die mit der Adresse der zu konfigurierenden CEL versehen sind. Eine KR muß dabei vollständig und korrekt auf eine Mehrzahl von CELs abgebildet werden, wobei mehrere CELs zu Gruppen zusammengefaßt sein können; die mit jeweils unterschiedlichen, aber vollständig ausgeführten KRs konfiguriert werden. Dabei sind alle CELs in einer Gruppe so verschaltet, daß nach Feststellung einer notwendigen Umkonfigurierung alle gruppierten CELs durch ein gemeinsames Signal (ReConfig) mitgeteilt wird, daß jede CEL die Datenverarbeitung zu beenden und in einen umkonfigurierbaren Zustand überzugehen hat.

Grundlagen der daedlockfreien Umkonfiguration

Bei zur Laufzeit umkonfigurierbaren Systemen tritt das Problem auf, daß das System in einen Zustand gelangen kann, in dem jeweils zwei Teile aufeinander warten und somit eine klassische Deadlock Situation eingetreten ist.

Dieses Problem könnte vermieden werden, indem eine neue Konfiguration immer nur ganz oder gar nicht in das System geladen wird, oder eine Art Timeout-Verfahren eingesetzt wird.

Dadurch entstehen ein Reihe von Nachteilen (benötigter Platz, Laufzeit etc.) und Problemen, wie zum Beispiel:

- Vorgehen, falls eine Konfiguration nicht geladen werden kann.
- Verwaltung der Reihenfolge, in der die Konfigurationen geladen werden
- Performance Einbruch, da andere Konfigurationen, welche eventuell in die CELs geladen werden könnten, nicht beachtet werden.

Mit dem folgend beschriebenen Verfahren können diese Probleme beseitigt werden. Es wird von einem DFP System nach DE 196 51 075.9 ausgegangen.

Von einer CEL aus, wird ein Trigger-Signal, wie in der DE 197 04 728.9 beschrieben, an eine CT gesendet. Diese CT stellt die Triggerquelle fest und wählt durch eine Look-Up Tabelle eine zu ladende Konfiguration (KR) aus. Die eingehenden Triggersignale werden gesperrt, es werden keine weiteren Trigger bis zur kompletten Abarbeitung der aktuellen Konfiguration akztepiert. Eine Konfiguration besteht aus mehreren Befehlen, welche an eine Menge von CELs übertragen wird. In einem zur Laufzeit konfigurierbaren System ist allerdings nicht sichergestellt, daß jeder Konfigurations-Befehl (KW) auch ausgeführt werden kann. Dies kann zum Beispiel daran scheitern, daß das adressierte konfigurierbare Element (CEL) seine Aufgabe noch nicht beendet hat und somit keine neuen Konfigurationsdaten entgegen nehmen kann. Um einen Performance Einbruch zu verhindern, werden alle Konfigurationsbefehle, welche nicht abgearbeitet werden konnten, entsprechend eines FIFOs hinter den letzten sich in einem – nachfolgend näher beschrieben – speziellen Speicher (FILMO) befindenden Konfigurationsbefehl geschrieben. Danach wird der nächsten Konfigurationsbefehl, nach dem gleichen Verfahren, abgearbeitet. Dies wiederholt sich solange, bis das Ende einer Konfiguration erreicht wurde.

Danach geht die CT, wieder in den Zustand über, in dem sie Trigger-Signale akzeptiert, um eventuell weiter Konfigurationen zu laden.

Eine Priorisierung der zu ladenden Konfigurationen wird erreicht, indem die CT den Speicher FILMO durchläuft, bevor die eigentlich zu ladende Konfiguration bearbeitet wird. Durch eine FIFO-ähnliche Struktur des FILMO wird sichergestellt, daß KW, welche während vorhergehenden Triggeranforderungen nicht vollständig abgearbeitet werden konnten, automatisch eine höhere Priorität vor den neu abzuarbeitenden WK erhalten. Bei der Abarbeitung des Speichers (FILMO) wird jedes durch einen Konfigurationsbefehl adressierte konfigurierbare Element (CEL) zuerst getestet, ob es sich im Zustand "umkonfigurierbar" befindet. Ist dieser Zustand "umkonfigurierbar", werden die Daten übertragen und aus dem Speicher FILMO gelöscht. Ist der Zustand "nicht umkonfigurierbar", bleiben die Daten im FILMO und werden beim nächsten Durchlauf erneut abgerarbeitet. Die CT verarbeitet den nächsten Eintrag im FILMO. Dies wiederholt sich solange, bis das Ende des FILMO erreicht ist. Danach wird die eigentliche, durch das Auftreten des Trigger-Signals aktivierte Konfiguration abgearbeitet. Der Aufbau des FILMOs entspricht dabei dem FIFO Prinzip, das heißt, es werden die ältesten Einträge zuerst verarbeitet.

Die übrigen, nicht beteiligten konfigurierbaren Elemente arbeitet während dieser Phase parallel weiter und wird nicht in ihrer Funktion beeinflußt. Dadurch kann der Fall eintreten, daß während die CT den FILMO abarbeitet, eine oder mehrere konfigurierbaren Elemente (CELs) in den Zustand "umkonfigurierbar" übergehen. Da die CT sich mit der Abarbeitung an einer beliebigen Stelle innerhalb des FILMOs befinden kann, könnte folgender Fall eintreten: Die CT versucht einen ersten Befehl abzuarbeiten dessen adressiertes konfigurierende Element (CEL) sich nicht in dem Zustand "umkonfigurierbar" befindet. Die CT fährt somit mit dem nächsten Befehl (KW) fort. Zur selben Zeit gehen ein oder mehrere konfigurierbaren Elemente in den Zustand "umkonfigurierbar" über, darunter auch das konfigurierbare Element, welches durch den ersten Konfigurationsbefehl hätte beschrieben werden können. Die CT verarbeitet einen zweiten Konfigurationsbefehl (KW), welcher das gleiche konfigurierbare Element (CEL) benutzt, wie der erste Konfigurationsbefehl, allerdings aus einer anderen Konfiguration stammt. Zu diesem Zeitpunkt, befindet sich das konfigurierbare Element (CEL) in dem Zustand "umkonfigurierbar" und der Befehl kann erfolgreich abgerarbeitet werden.

Durch diesen Ablauf ist nicht mehr sichergestellt, daß die Konfiguration, welche zuerst geladen werden sollte, auch tatsächlich zuerst fertiggestellt wird. Es können nun zwei teilweise fertige Konfigurationen existieren, welche jeweils konfigurierbare Elemente der anderen Konfiguration benötigen, um vollständig geladen zu werden. Eine klassiche Deadlock-Situation ist eingetreten.

In **Fig. 18** ist dieser Zustand dargestellt. Konfiguration A und Konfiguration B sollen konfiguriert werden. Die CT hat den schraffierten Teil von Konfiguration A und Konfiguration B bereits geladen. Konfiguration A benötigt zur Fertigstellung noch den hell-doppelt schraffierten Bereich von Konfiguration B, und Konfiguration B benötigt zur Fertigstellung noch den dunkel-doppelt schraffierten Bereich von Konfiguration A. Da beide Konfigurationen noch nicht vollständig abgeschlossen sind, und somit auch nicht funktionsfähig, tritt für keine der beiden Konfigurationen der Terminierungs-

zustand ein, in dem eine der beiden Konfigurationen entfernt würde. Beide Konfigurationen warten darauf, daß die noch benötigten konfigurierbaren Elemente freigegeben werden.

In dem vorliegenden Verfahren wird ein Deadlock verhindert, indem die CT vor der Abarbeitung des FILMOs die Zustände aller konfigurierbarer Elemente erfaßt und danach bis zur Beendigung des Vorgangs keine Änderungen mehr zuläßt, bzw. auftretende Änderungen ignoriert. Die CT arbeitet nur auf Basis der erfaßten Zuständen und nicht mit den aktuellen Zuständen der konfigurierbaren Elemente. Dadurch ist sichergestellt, daß jeder zu bearbeitende Befehl (KW) den gleichen Zustand der konfigurierbaren Elemente (CELs) vorfindet. Dieser Schritt schließt nicht aus, daß ein oder mehrere konfigurierbaren Elemente während der Abarbeitung des FILMOs, in den Zustand "umkonfigurierbar" übergehen. Diese Änderung ist für die CT während der Verarbeitung lediglich nicht sofort sichtbar, sondern erst zu Beginn des nächsten Durchlaufs.

Um die Problematik der Umkonfiguration nochmals zu verdeutlichen wird folgendes Beispiel gegeben:
Eine Matrix aus CELs ist unkonfiguriert und im RESET-Zustand. Jede CEL ist in der Lage anzuzeigen, ob sie sich in einem umkonfigurierbaren Zustand befindet. Alle CELs in der Matrix sind bereit konfiguriert zu werden; befinden sich also in einem umkonfigurierbaren Zustand. Eine erste Konfigurationsroutine (KR1) wird geladen, wobei die Matrix nicht vollständig benutzt wird. Die konfigurierten CELs heben die Anzeige, daß sie sich in einem konfigurierbaren Zustand befinden auf. In einen Teil der noch nicht konfigurierten CELs wird eine zweite, von der Ersten unabhängigen, Konfigurationsroutine (KR2) geladen. Eine dritte Konfiguration kann nicht geladen werden, da diese CELs der ersten und/oder zweiten Konfigurationsroutine (KR3) benötigt, die sich aber in keinem umkonfigurierbaren Zustand befinden, da sie benutzt werden.

KR3 muß so lange angehalten werden, bis die benötigten CEL freigegeben wurden, d. h. KR1 und KR2 terminiert haben. Während der Ausführung von KR1 und KR2 kommt eine Ladeanforderung für eine vierte Konfigurationsroutine (KR4) und eine fünfte Konfigurationsroutine (KR5) hinzu, die alle nicht sofort geladen werden können, da sie CELs benutzen, die von KR1 und KR2 verwendet werden. KR3 und KR4 benutzen teilweise die selben CELs, KR5 benutzt keine der CELs von KR3 und KR4.

Um KR3–5 ordentlich nachzuladen existieren folgende Forderungen:

1. KR3–5 sollen so geladen werden, daß die zeitliche Reihenfolge gemäß den Ladeanforderungen möglichst beibehalten wird.
2. Möglichst viele KR die unabhängig von einander sind, also keine gemeinsamen CELs besitzen, sollen geladen werden, um ein Höchstmaß an Parallelität zu erhalten.
3. Die KRs dürfen sich nicht gegenseitig blockieren, d. h. KR3 ist teilweise geladen, kann jedoch nicht weiter geladen werden, da andere CELs durch die teilweise geladene KR4 blockiert sind; während KR4 auch nicht weiter geladen werden kann, da wiederum benötigte CELs durch KR3 blockiert sind. Dies führt zu einer typische Deadlock-Situation.
4. Dem Compiler, der die KRs generiert hat ist es nicht möglich das zeitliche Zusammenspiel der KRs zu erkennen und so aufzulösen, daß es zu keiner Konfliktsituation kommt.

Dabei soll das Verhältnis zwischen den Aufwand für eine zu realisierende Schaltung und eines optimalen Ergebnisses möglichst gut sein, d. h. Ziel der Erfindung ist es mit möglichst geringem Aufwand eine flexible, parallele, Daedlock-freie Konfiguration zu ermöglichen, die mit wenig Zeit- und Rechenaufwand durchgeführt werden kann. Dabei müssen folgende Grundprobleme gelöst werden:

- Würde nur KR3 geladen werden, wäre das Verfahren Daedlock-frei, doch nicht optimal, da auch KR5 geladen werden könnte.
- Wird KR3 geladen, KR4 nicht, jedoch KR5 muß KR4 so vorgemerkt werden, daß es bei einem nachfolgenden Ladevorgang die höchste Priorität besitzt, was einen hohen Verwaltungsaufwand bedeutet.

Zur Lösung dieser und der Eingangs genannten Aufgaben dient die vorliegende Erfindung:

CACHE-PRINZIP

Die CT-Struktur ist hierarchisch aufgebaut, d. h. es existieren in einem Baustein mehrere CT-Ebenen. Die Anordnung entspricht vorzugsweise einer Baumstruktur (CT-Tree). Dabei ist der Wurzel-CT (Root-CT) der externe Konfigurationsspeicher (ECR) zugeordnet, während den Blättern die konfigurierbaren Elemente (CELs) zugeordnet sind. Den CTs der mittleren Ebenen sind jeweils die konfigurierbaren Elemente zugeordnet, die sich auf derselben Hierarchiestufe befinden.

Jeder CT ist ein lokaler interner Speicher zugeordnet. Dieser Speicher wird nur gelöscht, wenn neu zu speichernde KRs keinen Platz mehr haben. Dabei erfolgt das Löschen KR-weise, anhand einer Löschstrategie, so daß bestenfalls nur die KR gelöscht werden, die nicht mehr angefordert werden. Ebenfalls werden die KR einzeln gelöscht, nur genau so viele, daß genau so viel Speicher frei ist, wie notwendig ist um die neu zu ladende KR in den Speicher zu schreiben. Dadurch wird erreicht, daß möglichst viele KR in dem Speicher verbleiben.

Der Vorteil liegt darin, daß jede, einer beliebigen CTx untergordente CT, die sich also weiter oberhalb im CT-Baum befindet eine KR, die in der CTx gespeichert ist, nicht von dem externen Konfigurationsspeicher ECR anfordert, sondern direkt von CTx erhält. Dadurch ergibt sich eine Cachestruktur über mehrere Ebenen. Dadurch wird der Übertragungsaufwand im CT-Baum und ins besondere die benötigte Speicherbandbreite des ECR erheblich gesenkt. Auf Basis dieser Struktur ergeben sich auch mögliche Löschstrategien, die allerdings je nach Anwendung empirisch festgelegt werden sollten. Eine Möglichkeiten sind:

- Löschen des ältesten Einträge
- Löschen der kleinsten Einträge
- Löschen der größten Einträge
- Löschen der am seltensten abgerufenen Einträge.

Der Grundaufbau einer CT

Die nachfolgende Übersicht über die CT gibt einen Überblick über die einzelnen Baugruppen. Die detailierte Beschreibung der Baugruppen wird im Folgenden gegeben.

Kern einer CT ist die Steuer-Statemachine (CTS) die sämtliche Abarbeitungen von Konfigurationsroutinen (KRs) steuert. Der CTS zugeordnet ist, der Garbage-Kollektor (GC), der das Entfernen von KR aus dem Speicher (CTR) der CT steuert; das FILMO, das die Verwaltung der noch abzuarbeitenden KWs übernimmt und die LOAD-Statemachine, die das Laden von KRs steuert.

Der Speicher (CTR) ist als gewöhnlicher Schreib-Lese-Speicher ausgestaltet, wobei alle technisch möglichen Implementierungen zum Einsatz kommen können, und wird zur lokalen Speicherung von KRs für die jeweilige CT und deren untergeordnete CTs verwendet. Als Sonderfall kann der Speicher (CTR) auch als ROM, EPROM, EEPROM, Flash-ROM o. ä. ausgestaltet sein, um den Baustein mit einer festen, ASTC oder PLD-ähnlichen (siehe Stand der Technik) Funktion zu versehen.

Zur Generierung der CTR-Adressen werden vier als ladbare Zähler ausgestaltete Pointer verwendet:

1. Free-Pointer (FP): Zeigt auf den ersten freien Speicherplatz hinter der letzte KR im CTR.
2. Garbage-Pointer (GP): Zeigt auf einen durch den Garbage-Kollektor (GC) zu entfernenden Eintrag aus dem CTR.
3. Move-Pointer (MP): Zeigt auf eine Speicherstelle im CTR, von der ein gültiges, nicht zu entfernendes Konfigurationswort (KW), also einen Eintrag eines KR, an den durch GP definierten Eintrag kopiert/bewegt wird.
4. Program-Pointer (PP): Zeigt auf das momentan von der CTS ausgeführten KW.

KW werden über ein Ausgabe-Interface (OUT) an die zugehörenden CELs weitergegeben. Die CELs quittieren, sofern sie sich in einem umkonfigurierbaren Zustand befinden den Empfang der KW. Wird ein KW nicht quittiert, wird es in einem FIFO-ähnlichen Speicher (FILMO), zeitweise zwischengespeichert, um zu einem späteren Zeitpunkt, ohne den Program-Pointer zu benutzen, erneut an die adressierte CEL geschrieben zu werden.

Eine Aufforderung zur Abarbeitung eines KR erhält die CTS durch Triggersignale. Die Triggersignale durchlaufen eine Maske, das ist ein Filter, der unerwünschte Trigger ausfiltert (ausmaskiert). Eine Maske kann nach dem Stand der Technik durch UND-Gatter (AND) aufgebaut werden, die einen Trigger mit einem Freigabe-Signal UND-verknüpft. Die Trigger werden über einen priorisierten Round-Robin-Arbiter (SCRR-ARB) in Binärsignale umgewandelt. Ein priorisierter Round-Robin-Arbiter verknüpft den Vorteil der Gleichberechtigung eines Round-Robin-Arbiters mit der Erkennung der nächsten Freigabe in einem Takt, also dem Vorteil eines Prioritäts-Arbiter.

Die maskierten Trigger werden als Adresse auf eine erste Lookup-Tabelle (LUT1) geschaltet, das ist ein Speicher, der dem als Adresse eingehenden Trigger das ID der betreffenden KR zuordnet und auf den Datenleitungen ausgibt.

In einer zweiten Lookup-Tabelle (LUT2) wird die ID der KR der Adresse des Speicherplatzes der KR im CTR zugeordnet. Die zweite Lookup-Tabelle wird nicht nur zur Zuordnung von Trigger-Signalen verwendet, vielmehr benutzen einige Befehle, die einen ID als Parameter verwenden die LUT2 ebenfalls zur Adresszuordnung.

Die Zuordnung der Trigger-Signale zu den betreffenden IDs wird über den nachfolgend beschriebenen Befehl "REFERENCE" in die LUT1 eingetragen. Die Verwaltung der LUT2, also die Zuordnung der IDs zu den Adressen im CTR, geschieht automatisch durch die CTS und den GC.

Zum besseren Verständnis der CT ist im Folgenden ein möglicher Grundbefehlssatz dargestellt:

1. BEGIN <ID>

Durch BEGIN <ID> wird der Anfang einer Konfigurationsroutine gekennzeichnet. <ID> gibt die eindeutige Identifikationsnummer der Konfigurationsroutine an.

2. STOP

Durch STOP wird das Ende einer Konfigurationsroutine gekennzeichnet. An dieser Stelle beendet die Konfigurationstabelle (CT) die Abarbeitung der Konfigurationsroutine. Der Garbage-Kollektor (GC) beendet das Entfernen von Einträgen dieser Konfigurationsroutine.

3. EXECUTE <ID>

Springt zum Beginn (BEGIN <ID>) einer Konfigurationsroutine. Ist diese Routine nicht im Speicher der CT vorhanden, so wird sie von der darüberliegenden CT angefordert, bzw. aus dem Speicher geladen.

4. LOAD <ID>

Fordert die KR <ID> von der darüberliegenden CT an.

5. REMOVE <ID>

Ruft den GC auf, um die Konfigurationsroutine <ID> von BEGIN <ID> bis STOP aus dem Speicher der CT zu entfernen und die nachfolgenden Konfigurationsroutinen so weit vorzuschieben, daß kein Speicherloch durch die entfernte Konfigurationsroutine entsteht.

6. PUSH <FORCED> <ADDRESS> <DATA>

Schreibt die Konfigurationsdaten <DATA> an das Register <ADDRESS>. Ist <FORCED> gesetzt, werden die Daten auch geschrieben, wenn das RECONFIG-Flag des betreffenden Zielregisters nicht gesetzt ist.

7. MASK <SR> <TRIGGER>

Setzt die Trigger-Maske mit <TRIGGER>, bzw. setzt sie mit <TRIGGER> zurück, abhängig von <SR> (Set/Reset)

8. WAIT <UNMASCELD> <TRIGGER>

Hält die Abarbeitung der Konfigurationsroutine an und wartet auf den Trigger <TRIGGER>. Ist <UNMASCELD> gesetzt, wird auf das erwartete Trigger unabhängig des Zustandes der Trigger-Maske reagiert.

9. TRIGGER <TRIGGER>

Sendet den Binärwert eines Triggers an die übergeordnete CT.

10. REFERENCE <TRIGGER><ID>

Schreibt in die LUT1 bei Adresse <TRIGGER> den Wert <ID>, wodurch einem Triggersignal eine bestimmte KR zugeordnet wird.

Die Befehle EXECUTE, LOAD, REMOVE, PUSH, MASK, WAIT, TRIGGER, REFERENCE sind nur innerhalb der Klammer BEGIN . . . STOP gültig. Außerhalb dieser Klammer werden die Befehle nicht ausgeführt.

Der Aufbau einer Konfigurationsroutine (KR) sieht wie folgt aus:

BEGIN <ID>;
. . .
gültige Befehle
. . .
STOP;

Indirekte Addressierung (Referenzierung)

Das Cache-Prinzip der CT ermöglicht das Zwischenspeichern einer KR in einer CT, wobei die KR von mehreren unterschiedlichen tieferliegenden CTs oder CELs genutzt werden.

Werden von den tieferliegenden Einheiten Zugriffe auf das externe Interface des Bausteines (z. B. RAM, Peripherie) durchgeführt, ergibt sich die Notwendigkeit unterschiedliche Adressen oder Teile des externen Interfaces zu speichern. Dadurch würde sich der Inhalt der einzelnen benötigten KRs grundlegend unterscheiden. Ein Caching ist nicht mehr möglich.

Abhilfe schafft eine indirekte Referenzierung. Dazu werden spezielle KR (im folgenden IKR genannt) verwendet, die die notwendigen externen Paramter beinhalten und setzen. Eventuell werden über Trigger andere unterschiedliche KRs in verschiedenen Hierarchieebenen aufgerufen. Ab Ende einer IKR wird das eigentliche KR aufrufen. Lediglich die IKR sind nicht cachebar, während die aufgerufenen KR durchaus einheitlich und daher cachebar sind. Es ist sinnvoll, die Größe der IKR auf das absolute Minimum zu reduzieren, nämlich ausschließlich die externen und unterschiedlichen Parameter und den Aufruf der einheitlichen KR. Eine indirekte Konfigurationsroutine (IKR) ist wie folgt aufgebaut:

BEGIN <TD>;
. . .
xxx; gültige Befehle, wobei lediglich externe Peripherie angesteuert werden sollte,
TRTGGER <ID>; Start-, Stop- oder Lade-Anforderungen an Periphere Prozesse
. . .
GOTO <ID>; Sprung zur einheitlichen KR STOP;

Sonderfälle

1. WATT_FOR_BOOT

Dieses Kommando ist nur an der ersten Adresse des CTR gültig. Während des Boot-Vorganges wird zunächst die komplette Boot-KR in das CTR geschrieben, jedoch nicht die Beginnsequenz des Boot-KR BEGIN <0>. An dessen Stelle (auf Adresse 1) steht WAIT-FOR-BOOT, das bei einem RESET automatisch gesetzt wird. Erst nachdem die gesamte Boot-KR in das CTR geschrieben ist, wird WATT_FOR_BOOT mit BEGIN <0> überschrieben und die CTS beginnt mit der Abarbeitung der Boot-KR.

WATT_FOR_BOOT darf nicht innerhalb eines Programmes auftreten.

2. BOOT <CT-ID>

BOOT <CT-ID> kennzeichnet in welche CT die nachfolgende Boot-KR geschrieben werden soll. Nach BOOT <CT-ID> folgt kein BEGIN, die Boot-KR wird nicht durch STOP, sondern durch ein nachfolgendes BOOT <CT-ID> abgeschlossen. Ein STOP beendet den Bootvorgang.

BOOT <CT-ID> darf nicht innerhalb eines Programmes auftreten.

Boot-Vorgang

Nach einem RESET lädt die CT des obersten Hierarchie-Levels (ROOT-CT) die Boot-KR in die CTs der unteren Hierarchien. Dazu existiert ein Sprung an eine festgelegte Adresse (BOOT-ADR) im, der ROOT-CT zugeordneten, externen Konfigurationsspeicher (ECR). Die ROOT-CT führt diesen Sprung durch und erreicht die Boot-Sequenz. Diese ist wie folgt aufgebaut:

BOOT <CT-ID0>; COMMAND; COMMAND; . . .
BOOT <CT-ID1>; COMMAND; COMMAND; . . .
. . .
BOOT <CT-IDn>; COMMAND; COMMAND; . . .
STOP;

Während des Boot-Vorganges wird zunächst die komplette Boot-KR in das CTR ab Adresse 2 der durch <CT-ID> angegebenen CT geschrieben. Die Beginnsequenz des Boot-KR (BEGIN <0>) wird nicht auf Adresse 1 geschrieben. An dessen Stelle steht WAIT-FOR-BOOT, das bei einem RESET automatisch gesetzt wird. Erst nachdem die gesamte Boot-KR in das CTR geschrieben ist, und die ROOT-CT das nächste BOOT <CT-ID> erreicht hat, wird STOP an das Ende des Boot-KR in das CTR geschrieben und WAIT_FOR_BOOT mit BEGIN <0> überschrieben. Die CTS beginnt mit der Abarbeitung der Boot-KR.

Laden einer Konfigurationsroutine

Es existierem drei Gundmechanismen um eine Konfigurationsroutine, außer der Boot-KR anzufordern:

1. Ausführen eines LOAD <ID> durch die CTS
2. Ausführen eines EXECUTE <ID> durch die CTS, wobei die KR mit der betreffenden ID nicht im CTR vorhanden ist.
3. Auftreten eines Triggers, der über die LUT1 auf einen <ID> übersetzt wird, dessen zugehörige KR nicht im CTR vorhanden ist.

Der Ablauf in allen drei Fällen ist derselbe:
Die ID der angeforderten KR wird der LUT2 als Adresse angegeben. Die LUT2 überprüft, ob eine gültige Adresse im CTR existiert. Existiert diese nicht, d. h. <ID> zeigt in der LUT2 auf den Wert 0, wird load <ID> an die CTS gesendet.

Die CTS fordert daraufhin die <ID> betreffende KR bei der hierarchisch übergeordneten CT an. Diese Anforderung erreicht die übergeordnete CT in Form eines Triggers und wird entsprechend von ihr ausgewertet.

Die übergeordnete CT sendet die angeforderte KR an die anfordernde CT. Die Daten werden ab der Adresse, auf die der FREE-POINTER (FP) zeigt in das CTR geschrieben, wobei der FP nach jedem Schreibzugriff um eins erhöht wird. Erreicht der FP die obere Grenze des CTR, wird der Garbage-Kollektor (GC) aufgerufen, um die unterste KR innerhalb des CTR zu entfernen und das CTR zu komprimieren. Der FP wird dabei neu gesetzt. Dieser Vorgang findet so lange statt, bis die zu ladende KR komplett in das CTR paßt.

Sprungtabelle im Konfigurationsspeicher

Der der ROOT-CT zugeordnete Konfigurationsspeicher beinhaltet sämtliche KR, die für eine Applikation geladen werden müssen. Im externen Konfigurationsspeichers (ECR) befindet sich an einer festgelegten Adresse (ADR-BOOT) Sprung zu der Boot-Konfigurations-Routine. In einem weiteren festgelegten Speicherbereich (LUT-ECR) beliebiger, jedoch fest vorgegebener Länge die Sprünge zu den einzelnen KRs. Dabei wird die <ID> der jeweiligen KR als Adresse im ECR verwendet, an der die Startadresse der jeweiligen KR steht; wodurch KRs indirekt adressiert werden:

ID → LUT-ECR → KR

Änderung der KR im Konfigurationsspeicher

Die KR mit der ID <A> soll geändert werden. Zunächst schreibt der HOST die neue KR für die ID <A> an eine freie Speicherstelle im ECR. Die ID <A> wird zusammen mit der neuen Adresse der KR im Konfigurationsspeicher von der übergeordneten Einheit (HOST) in ein dafür vorgesehenes Register der ROOT-CT geschrieben. Die ROOT-CT sendet an alle darunterliegenden CTs das Kommando REMOVE <A>. Daraufhin entfernen alle CTs beim Erreichen eines STOP oder während IDLE-Zyklen, also sobald keine KR ausgeführt wird, die auf diese ID bezogene KR aus dem CTR und setzen die LUT2 an Adresse <A> auf "NoAdr", das bedeutet, es existiert keine gültiger Adresseintrag für ID <A> in LUT2. Wird die ID <A> erneut angefordert, zwingt der fehlende Eintrag ("NoAdr") an Stelle <A> in die LUT2 jede CT die KR <A> vom ECR neu anzufordern.

Das FILMO

Ein KR besteht hauptsächlich aus dem Befehl PUSH, der neue Konfigurationsworte an eine bestimmte Adresse schreibt. Ist das Schreiben eines Konfigurationswortes nicht möglich, da das adressierte konfigurierbare Element (CEL) nicht bereit ist eine neue Konfiguration zu empfangen, wird das Konfigurationswort statt an das adressierte konfigurierbare Element (CEL) in einen Speicher, im folgenden FILMO genannt, geschrieben. Die nachfolgenden Befehle werden normal abgearbeitet, bis erneut ein Konfigurationswort nicht geschrieben werden kann, das dann in das FILMO geschrieben wird.

Das FILMO wird in IDLE-Zyklen und vor jedem Ausführen eines neuen KR komplett durchlaufen. Dabei wird, beginnend beim ältesten Datenwort, entsprechend eines FIFOs nach dem Stand der Technik, jedes ausgelesene Wort des FILMOs an sein adressiertes Element zu senden; dabei muß das adressierte Element bereit sein das Konfigurationswort zu empfangen. Sofern die Datenwörter von Beginn an geschrieben werden können (d. h. die adressierten konfigurierbaren Elemente (CELs) sind bereit) wird der Eintrag aus dem FILMO nach Art eines FTFOs entfernt. Kann ein Konfigurationswort nicht geschrieben werden, wird es übersprungen und nicht aus dem FILMO entfernt. Im Gegensatz zu einem FIFO werden die Daten nach dem übersprungenen Konfigurationswort weiter ausgelesen. Konfigurationsworte, die nach einem übersprungenen Konfigurationswort geschrieben werden können werden entweder je nach Implementierung des FILMOs

1. als geschrieben markiert und nicht aus dem FILMO gelöscht, wobei als geschrieben markierte Konfigurationswörter bei den folgenden Durchläufen nicht mehr gelesen werden, bzw. sofort gelöscht werden, sofern kein übersprungenes Konfigurationswort mehr vor ihnen liegt; oder
2. aus dem FILMO gelöscht, wobei die Konfigurationswörter vor und nach dem gelöschten Konfigurationswort erhalten bleiben, dabei müssen zum Löschen die nachfolgenden Worte nach vorne (oben) oder die davorliegenden Worte nach hinten (unten) geschoben werden, wobei die Reihenfolge der Konfigurationsworte unbedingt beibehalten wird.

Wird eine neue KR ausgeführt, werden die Konfigurationsworte (KW), die von der CTS nicht an die adressierten Elemente (CELs) geschrieben werden konnten, erneut an das FILMO angehängt, d. h. die KW werden an das Ende (aus Leserichtung) des FILMOs geschrieben. Ist das FILMO voll, d. h. es existieren keine freien Einträge für Konfigurationsworte, wird die Ausführung des KR gestoppt. Das FILMO wird so lange durchlaufen, bis genügend Konfigurationsworte geschrieben werden konnten und entsprechend viele freie Einträge entstanden sind, woraufhin das KR weiter abgearbeitet wird.

Das FILMO stellt einen FIFO-ähnlichen Speicher dar, der immer vom ältesten Eintrag an linear durchlaufen wird, im Gegensatz zu einem FIFO werden jedoch Einträge übersprungen (First In Linear Multiple Out)

Die Funktion der Konfigurationstabellen-Statemachine (CTS)

Die Konfigurationstabellen-Statemachine (CTS) übernimmt die Steuerung der CT. Dabei führt sie die Befehle der KR aus und reagiert auf eingehende Trigger. Sie übernimmt die Verwaltung des FILMOs, i.b. liest sie in IDLE-Zyklen und vor dem Ausführen einer KR das FILMO aus.

Sie reagiert auf die von der LUT-Struktur generierten Signalen illegal <ID> (Illegal Trigger, siehe **Fig.** 1, **0102**) und load <ID>. load <ID> wird generiert, wenn ein Cache-Miss in LUT2 vorliegt (**0105**), oder die durch ID referenzierte KR/IKR als gelöscht markiert wurde (**0107**) Sie reagiert auf die Steuersignale der übergeordneten CT. Ein Implementationsbeispiel für die Verarbeitung der Befehle ist in den **Fig.** 2 bis 7 dargestellt.

Steuersignale an übergeordnete CTs

illegal <ID> (**0102**)
load <ID> (**0105/0107**)
trigger <trigger> (**0108**)

Steuersignale von übergeordneten CTs

remove <ID> (siehe **Fig.** 15, **1513**)
write_to_FP <data> (siehe **Fig.** 2, **0205**).

Die Funktion des Garbage-Kollektors (GC)

Der CTR unterliegt zwei Problemen:

1. Verweist ein LOAD- oder EXECUTE-Befehls, bzw. ein Trigger, auf eine ID, deren KR nicht im CTR vorhanden ist, muß die KR nachgeladen werden. U.U. ist jedoch nicht genügend Platz im CTR vorhanden um die angeforderte KR zu laden.
2. Beim Auftreten eines REMOVE <ID> ist die entsprechende KR aus dem CTR zu entfernen. Dabei entsteht, sofern sich dike KR nicht am Ende des CTR befindet eine Lücke. Beim Laden einer neuen KR wird die Lücke u. U. nicht wieder ganz aufgefüllt oder die Lücke ist zu klein für die neue KR. Dies führt zu einer Fragmentierung des CTR. Die Aufgabe des Garbage-Kollektor ist es, KR aus dem CTR zu entfernen, um Platz für neue Einträge zu schaffen UND nach Entfernen der Einträge den CTR so umzuorganisieren, daß alle verbleibenden KR als geschlos-

sener Block hintereinander im Speicher liegen und die freigewordenen Speicherblöcke als ein geschlossener Block an einem Ende des CTR liegen. Dadurch können auf optimale Weise und ohne Verluste an Speicherplatz neue KR nachgeladen werden.

Auswerten von Triggerimpulsen

Jede CT besitzt einen Anschluß an mehrere zu ihrer jeweiligen Hierarchieebene gehörenden Triggersignale, die zu einem Bus zusammengefaßt sind. Eingehende Trigger werden über eine Maske ausgewertet, d. h. nur die freigeschalteten Triggersignale werden weitergeleitet. Die freigeschalteten Triggersignale werden taktsynchron in einem Sample-Register zwischengespeichert (gesampled). Ein Round-Robin-Arbiter wählt ohne Priorität eines der gespeicherten Triggersignale aus und wandelt das Signal in einen binären Vektor. Das gewählte Triggersignal wird aus den Sample-Register gelöscht. Der Binärvektor wird an eine erste Lookup-Tabelle (LUT1) weitergeleitet, die den Binärvektor in die Identifikationsnummer (ID) der aufzurufenden Konfigurationsroutine (KR) übersetzt. Die ID wird in einer zweiten Lookup-Tabelle (LUT2) in die Adresse der KR im CT-Speicher (CTR) übersetzt. Die CT-Statemachine (CTS) setzt ihren Programm-Pointer (PP) auf diese Adresse und beginnt mit der Ausführung der KR. Voraussetzung ist, daß jeder über die Maske freigeschaltete Trigger einen entsprechenden Eintrag in LUT1 besitzt. Fehlt dieser, wird ein Fehlerzustand an die CTS weitergeleitet (illegal trigger), dabei wird jede ID = "NoAdr" als nicht vorhandener Eintrag gewertet. "NoAdr" ist ein implementationsabhängig gewähltes Token.

Fehlt der Eintrag in LUT2, d. h. die auf die ID bezogene KR befindet sich nicht im CTR, wird eine Ladeanforderung an die CTS gesendet (load <ID>).

Senden von Triggerimpulsen an die übergeordnete CT Neben der bereits beschriebenen Schnitt stelle zu einer übergeordnerten CT zum Laden von KR existiert eine weitere Schnittstelle zum Austauschen von Triggersignalen. Dabei sendet eine CT

1. an alle anderen CTs einen Triggervektor (BROADCAST)
2. an die darüberliegende CT einen Triggervektor (HIGHER)
3. an die darunterliegende CT einen Triggervektor (LOWER)
4. an eine beliebige adressierte CT einen Triggervektor (ADDRESSED)

Wobei ein Triggervektor einen Binärwert darstellt, der auf einen Eintrag in der LUT2 der empfangenden CT referenziert. Der Mechanismus ist notwendig um beispielsweise innerhalb einer IKR eine KR in einer weiteren CT zu starten um beispielsweise die Peripherie oder den Speicher anzusteuern.

Zur Weiterleitung von Triggersignalen an eine übergeordnete CT existieren 2 Mechanismen:

1. Der LUT1 wird ein Bit hinzugefügt, das angibt, ob der Inhalt des Speichers als KR ID oder als Binärwert für einen Triggerimpuls betrachtet wird. Liegt ein Triggerimpuls vor, wird der Dateninhalt von LUT1 direkt als Trigger an die übergeordnete CT gesendet.
2. Mit dem Befehl TRIGGER kann der Binärwert eines Triggers angegeben werden, der direkt an die übergeordnete CT gesendet wird.

Der priorisierte Round-Robin-Arbiter

Der priorisierte Round-Robin-Arbiter (Single-Cycle-Round-Robin-Arbiter SCRR-ARB) ist taktsynchron aufgebaut, d. h. bei jeder – je nach Implementierung positiven oder negativen – Taktflanke (TF1) liefert er ein Ergebnis. Die eingehenden Signale (ARB-IN) durchlaufen eine Maske (ARB-MASK), die von dem Arbiter gemäß dem nachfolgend beschriebenen Verfahren selbst verwaltet wird. Die Ausgangssignale der Maske werden an einen Prioritätsarbiter nach dem Stand der Technik geleitet. Der Arbiter liefert taktsynchron bei jeder Taktflanke (TF1) ein Ergebnis (ARB-OUT), d. h. den Binärwert des höchstpriorisierten Signals nach der Maske (ARB-MASK). Dem Ergebnis zugeordnet ist ein Signal (VALID), das angibt, ob der Binärwert gültig oder ungültig ist. Abhängig von der Implementierung der Prioritätsarbiters ist es möglich, daß beim Anliegen des Signals 0 und beim Anliegen keines Signals derselbe Binärwert generiert wird: In diesem Fall zeigt VALID an, daß das Ergebnis ungültig ist, sofern kein Signal anliegt. Dieses Signal wird

1. als Ergebnis der Arbiters ausgegeben und
2. auf einen Dekoder geschaltet, der die Binärwerte – wie in der folgenden Tabelle beispielsweise für einen 3-bit Binärwert angeben – auskodiert. (Das Kodierungsverfahren ist gemäß dieses Prinzips auf jeden beliebigen Binärwert anpaßbar):

Binärwert (ARB-OUT)	Auskodierung (ARB-DEC)	Bemerkung
111	0111 1111	
110	0011 1111	
101	0001 1111	
100	0000 1111	
011	0000 0111	
010	0000 0011	
001	0000 0001	
000	1111 1111	Reset-Zustand und wenn Binärwert (ARB-OUT) ungültig

Dem Dekoder zugeordnet ist ein Register (ARB-REG), das die auskodierten Werte (ARB-DEC) des Dekoders bei der zu TF1 inversen Taktflanke (TF2) übernimmt. ARB-DEC wird auf die Maske (ARB-MASK) zurückgekoppelt und schaltet die einzelnen Eingangssignale (ARB-IN) frei.

Der funktionale Ablauf im Arbiter ist wie folgt:

1. Nach einem RESET sind alle ARB-IN über ARB-MASK freigeschaltet, da ARB-DEC alle Signale auf "Freigabe" stellt.
2. Das höchst priorisierte gesetzte ARB-IN (beispielsweise besitzt in der obigen Tabelle das Signal 7 (binär 111) die höchste Priorität und 0 (binär 000) die niederste Priorität) wird als Binärwert ausgegeben.
3. Über ARB-DEC wird das Signal gesperrt, sowie alle weiteren Eingänge die evtl. noch höher priorisiert wären, aber nicht gesetzt sind.
4. Die folgenden Schritte 5 und 6 wiederholen sich so lange, bis das Signal 0 (binär 000) erreicht ist, oder kein Signal hinter ARB-MASK mehr anliegt. Dann schaltet ARB-DEC (siehe Auskodierungstabelle) wieder alle Signale durch ARB-MASK über ARB-DEC frei und der Ablauf beginnt bei Schritt 2.
5. Das nunmehr höchst priorisierte gesetzte ARB-IN wird als Binärwert ausgegeben.
6. Über ARB-DEC wird das Signal gesperrt, sowie alle weiteren Eingänge die evtl. noch höher priorisiert wären, aber nicht gesetzt sind. (Weiter mit Schritt 4).

Dadurch wird erreicht, daß alle Eingangssignale gleichberechtigt behandelt werden und bei jedem Taktzyklus eines der Eingangssignale (ARB-IN) binär aus kodiert und ausgegeben (ARB-OUT) wird.

ARB-REG kann mit einem Enable-Eingang (EN) versehen werden, der eine Änderung des Registerinhaltes nur bei TF2 zuläßt, wenn ein entsprechendes Signal anliegt. Dadurch wird nicht bei jedem Takt ein Binärvektor ausgegeben, sondern abhängig von einer Freischaltung durch EN und TF2. Der Eingang wird zur Synchronisation notwendig, wenn die nachgeordnete Schaltung die Verarbeitung nicht in einem Taktzyklus durchführen kann, sondern mehrere Zyklen benötigt und erst dann den nächsten Binärvektor akzeptiert.

Figuren

Die nachfolgend beschriebenen Figuren verdeutlichen anhand eines Implementationsbeispiels die Verwaltung von Konfigurationsdaten nach dem vorgestellten Verfahren:

Fig. 1 Verfahren der Adressgenerierung innerhalb der Lookup-Tabellen
Fig. 2–7 Abarbeitung der Befehle und Funktion der Statemachinen
Fig. 8 Aufbau des SCRR-ARB
Fig. 9 Aufbau der LUT1 & LUT2
Fig. 10 Aufbau der Pointerarithmetik und des CTR
Fig. 11 Aufbau eines FILMO
Fig. 12a Hierarchische Anordnung der CTs
Fig. 12b Senden eines Triggers zwischen den CTs
Fig. 12c, d Methoden zum Senden eines
Fig. 13 Aufruf einer KR durch mehrere IKR
Fig. 14 Aufbau der LUT1 einer ROOT-CT
Fig. 15 Aufbau der HOST-Steuerung einer ROOT-CT
Fig. 16 Verdeutlichung des LUT und ECR Konzeptes
Fig. 17 Ablaufsteuerung einer CT mittlerer Hierarchieebene, bzw. einer ROOT-CT
Fig. 18 Deadlockproblematik bei der Konfiguration eines 2-dimensionalen Arrays
Fig. 19 Verdeutlichung des FILMO-Konzeptes

Beschreibung der Figuren

Fig. 1 zeigt den Ablauf der CTR-Adressgenerierung innerhalb einer CT. Dabei wird ein eingehender binärer Triggervektor (**0101**) in der LUT1 auf eine gültige KR oder IKR ID übersetzt. Existiert keine gültige ID, wird ein Signal "Illegal Trigger" generiert (**0102**), das anzeigt, daß der Trigger nicht in LUT1 bekannt ist. Das Signal kann als Fehlermeldung an die übergeordnete CT weitergeleitet oder ignoriert werden. Die Übersetzung von "Trigger" nach "ID" wird mittels des Befehls "REFERENCE" in die LUT1 eingetragen.

Eine gültige ID (**0103**) wird an die LUT2 weitergeleitet. IDs die innerhalb von Befehlen, also durch einen Operanden, angegeben sind (**0104**), treffen direkt auf die LUT2. Die LUT2 übersetzt eine eingehende ID in die Adresse der KR/IKR innerhalb des CTR. Ist die KR/IKR nicht im CTR gespeichert (es liegt im Cache nicht vor), wird das Signal "Miss" generiert (**0105**). Ist die übersetzte Adresse der KR/IKR mit dem Token "NoAdr" markiert, wird mit "NoEntry" (**0107**) angezeigt, daß die Adresse gelöscht ist. "Miss" und "NoEntry" zeigen an, daß eine Übersetzung auf eine CTR-interne Adresse nicht möglich ist. Auf Grundlage dieses Signals lädt die LOAD-Statemachine die KR/IKR mit der entsprechenden ID von einer darüberliegenden CT nach. Sofern eine gültige Adresse vorhanden ist, wird diese an die Pointerarithmetik des Adressgenerators weitergeleitet (**0106**). In LUT1 wird ein eingehender binärer Triggervektor entweder in eine ID oder einen weiteren Triggervektor übersetzt, wobei in diesem Fall der Triggervektor ausgegeben wird (**0108**).

In **Fig.** 2 ist der Ablauf beim Laden einer KR/IKR dargestellt. Zunächst wird die ID (**0201**) der zu ladenden KR/IKR an die darüberliegende CT gesendet. Daraufhin wird in die LUT2 an der Stelle des Eintrages für die angeforderte ID der Wert des FreePointers (FP) eingetragen. FP zeigt auf den Eintrag hinter dem letzten für eine KR/IKR genutzen Eintrag im CTR. Dies ist der erste Eintrag, auf den die zu ladende KR/IKR gespeichert wird.

Die Statemachine wartet auf ein Datenwort von der drüberliegenden CT. Sobald das Wort verfügbar ist, wird es an die durch FP referenzierte Stelle geschrieben. FP wird inkrementiert. Zeigt FP auf einen Eintrag hinter dem Ende des CTR wird der erste Eintrag im CTR entfernt um Platz zu schaffen (**0202**); dabei wird FP aktualisiert.

Ist das von der darüberliegenen CT gesendete Datenwort "STOP", wird der Ladevorgang abgebrochen (**0203**), ansonsten mit dem Warten auf ein neues Datenwort fortgesetzt (**0204**).

In **Fig.** 3a ist der "MASK"-Befehl dargestellt. Der Operand des Befehls wird in das MASK-Register geschrieben. Das MASK-Register befindet sich am Eingang der Triggersignale vor LUT1 und maskiert ungültige Trigger aus.

In **Fig.** 3b wird durch den Befehl "TRIGGER" der Operand des Befehls als Triggervektor zu den anderen CTs abgesendet.

In **Fig.** 3c wird durch den Befehl "REFERENCE" die Übersetzung eines Triggers zu der entsprechenden KR/IKR ID in die LUT1 geschrieben.

In **Fig.** 4a wir der Befehl "WAIT" dargestellt. Der Operand des Befehls wird in das WAITMASK-Register geschrieben. Alle Trigger, bis auf den/die Erwarteten und daher in WAITMASK freigeschalteten werden ignoriert. Erst nach Auftreten des Triggers wird zum Programmfluß zurückgekehrt.

In **Fig.** 4b ist der "PUSH"-Befehl abgebildet. Das Konfigurationswort wird zum adressierten konfigurierbaren Element (CEL) gesendet. Akzeptiert das CEL das Konfigurationswort nicht; da das CEL sich beispielsweise im Zustand "nicht konfigurierbar", (vgl. DE 197 04 728.9) befindet; wird das Konfigurationswort in den FILMO geschrieben (**0401**).

Fig. 5 zeigt den Ablauf eines "REMOVE"-Befehles. Es gibt zwei Aufrufvarianten:

1. Die erste im CTR liegende KR/IKR wird aus dem CTR entfernt. Dem GarbagePointer (GP) wird die Adresse 0 des CTR zugewiesen (**0501**).
2. Eine spezifisch durch ihre ID angegebene KR/IKR wird aus dem CTR entfernt. Dem GarbagePointer (GP) wird die erste Adresse des zu entfernenden KR/IKR im CTR zugewiesen (**0502**).

Der MovePointer wird mit dem Wert von GP geladen. GP und MP referenzieren auf einen "BEGIN <ID>"-Befehl im CTR, auch wenn die erste KR/IKR aus dem CTR entfernt werden soll. Die betreffende ID wird in LUT2 als ungültig markiert. MP wird so lange inkrementiert, bis das "BEGIN <ID>" des nächsten im Speicher liegenden KR/IKR erreicht wird (**0503**), ODER MP gleich dem FreePointer (FP) ist, das bedeutet, daß die zu entfernende KR/IKR die letzte im CTR ist (**0504**).

- In diesem Fall wird FP mit dem Wert von GP geladen, wodurch die durch die zu löschende KR/IKR belegten Speicherstellen als frei markiert werden; und die Funktion "REMOVE" ist beendet (**0505**).
- Andernfalls ("BEGIN <ID>" wird erreicht (**0506**)) werden die durch MP referenzierten Daten an die durch GP referenzierte Speicherstelle kopiert. MP und GP werden inkremetiert. Dieser Ablauf findet so lange statt, bis MP das Ende von CTR oder die Position von FP erreicht hat (**0507**). Wird während des Ablaufes durch MP eine Speicherstelle referenziert, in der "BEGIN <ID>" steht, wird der Eintrag für die entsprechende ID in LUT2 mit MP überschrieben (**0508**), damit bei einem Lookup die richtige Speicherstelle ausgegeben wird.

Fig. 6 zeigt das Ablaufdiagram des FILMOs. Ein FILMO beinhaltet drei Pointer:

1. WriteP: Der Schreibzeiger des FILMO-RAM
2. ReadP: Der Lesezeiger des FILMO-RAM
3. FullP: Der Zustandszeiger, der den "Füllstand" des FILMO-RAMs repräsentiert und einen Unterlauf, bzw. Überlauf verhindert.

Ein ein-Bit Register "BeginF" zeigt an, ob sich der aktuelle Lesezugriff am Anfang des FILMO-RAMs befindet (TRUE), d. h. keine nicht gelöschten Einträge befinden sich zwischen dem Lesezeiger und dem Beginn des FILMO-RAMs; oder sich der Lesezeiger in der Mitte des FILMO-RAMS befindet (FALSE), also benutzte Einträge zwischen

dem Lesezeiger und dem Beginn des FTLMO-RAMS liegen.

Weiterhin existieren zwei Register zum Speichern der Zustände des ReadP und FullP. Es ist notwendig beim Auftreten des ersten ungelöschten Eintrages die beiden Register zu sichern, da bei einem später stattfindenden Lesezugriff an der Stelle dieses Eintrages mit dem Auslesen begonnen werden muß. Andererseits müssen jedoch ReadP und FullP während des aktuellen Lesevorganges weiterhin modifiziert werden, um die nächsten Leseadressen zu erhalten, bzw. das Ende des FILMO-RAMs festzustellen. Durch den Aufbau des FTLMOs als FIFO-ähnliche Struktur – als sogenannten Ringspeicher – kann Beginn und Ende des Speichers nicht anhand einer Adresse 0 oder eine Maximaladresse festgelegt werden.

Aus dem Grundzustand führen zwei Ablaufpfade:

1. Lesepfad (0601)

FullP und ReadP werden in die Register gesichert.

Die Abarbeitungsschleife beginnt:

BeginF ist TRUE.

Ist FullP gleich 0, werden ReadP und FullP aus ihren Registern zurückgelesen (**0602**) und die Statemachine springt in den Grundzustand zurück.

Ansonsten (**0603**) wird getestet, ob der Eintrag im FILMO, auf den ReadP zeigt gleich "NOP" ist, d. h. es handelt sich um einen als gelöscht markierten Eintrag in der Mitte des FILMOs. Ist dies nicht der Fall (**0604**) wird versucht den Eintrag in das konfigurierbare Element (CEL) zu schreiben. Gelingt dies nicht (REJECT, vgl. DE 197 04 728.9, **0605**), da CEL nicht umkonfigurierbar ist, wird BeginF auf FALSE gesetzt, FullP dekrementiert und ReadP inkrementiert. Die Statemachine springt an den Beginn der Abarbeitungsschleife (**0606**). Gelingt das Schreiben des Eintrages an das CEL (**0607**), oder der Eintrag ist ein NOP, wird BeginF gestestet:

BeginF == TRUE (**0608**): Es liegen keine ungelöschten Einträge vor diesem. FullP wird inkrementiert, ReadP wird in dem zugeordneten Register gesichert, um den neuen Anfang des FILMOs festzuhalten. FullP wird gesichert um die aktuelle Datenmenge festzuhalten; ReadP wird inkrementiert.

BeginF == FALSE (**0609**): FullP wird inkrementiert und der aktuelle Eintrag im FILMO-RAM mit NOP überschrieben, d. h. der Eintrag wird gelöscht. ReadP wird inkrementiert. In beiden Fällen springt die Statemachine an den Beginn der Abarbeitungsschleife.

2. Schreibpfad (**0610**)

Es wird getestet, ob der FILMO-RAM voll ist, indem FullP auf den maximalen Wert überprüft wird. Ist dies der Fall (**0611**), wird in den Lesepfad gesprungen um Platz zu schaffen.

Ansonsten wird das Datenwort in den FTLMO-RAM geschrieben und WriteP und FullP inkrementiert.

Fig. 7 zeigt den Ablauf in der Hauptstatemachine. Der Grundzustand (IDLE) wird verlassen, sobald ein

1. REMOVE-Kommando von der darüberliegenden CT auftritt (**0701**):
   Der REMOVE-Befehl wird ausgeführt und die Statemachine kehrt nach IDLE zurück.
2. Ein Triggersignal zur Generierung eines Triggers zwischen den CTs auftritt (**0702**):
   Der Trigger wird ausgegeben. Die Statemachine springt in den "STOP"-Befehl und danach nach IDLE zurück.
3. Ein Triggersignal zur Ausführung eines KR/IKR <ID> auftritt (**0703**):
   Der ProgramPointer (PP) wird mit der durch LUT2 generierten Adresse geladen. Ist die Adresse ungültig, d. h. kein Eintrag für das zu ladende KR/IKR vorhanden, wird dieses geladen (**0704**) und PP neu gesetzt.

Die Ausführungsschleife beginnt:

PP wird inkrementiert (beim ersten Schleifendurchlauf wird dadurch der BEGIN <ID>-Befehl übersprungen), das Auftreten weiterer Trigger wird unterbunden, RECONFigur (vgl. DE 197 04 728.9) wird gesperrt. Die Befehle werden ausgeführt und zum Beginn der Ausführungsschleife gesprungen (**0707**).

Der Befehl "STOP" wird gesondert ausgeführt (**0705**). Die Trigger und RECONFIG (vgl. DE 197 04 728.9) werden wieder freigeschaltet und die Statemachine springt nach IDLE. Der Befehl "EXECUTE" wird ebenfalls gesondert ausgeführt (**0706**). Die in EXECUTE <ID> angegebene TD wird in das ID-REG geschrieben. PP wird neu geladen und die durch ID angegebene KR/IKR ausgeführt (**0708**).

Nach einem Reset der CT wird die Grundkonfiguration in das CTR geladen und direkt in die Ausführung der Grundkonfiguration gesprungen (**0709**)

Fig. 8 zeigt den Aufbau eines SCRR-ARB. Die zu arbitrierenden Signale gelangen über DataIn auf eine Maske (**0801**), die gemäß der bekannten Tabelle einen zusammenhängenden Teil der Signale durchschaltet, bzw. sperrt. Ein gewöhnlicher Prioritätsarbiter (**0802**) nach dem Stand der Technik arbitriert ein Signal aus der Menge der Durchgeschalteten und lieferte dessen Binärvektor (BinaryOut) zusammen mit einer gültig/ungültig-Kennung (ValidOut) (ebenfalls gemäß dem Stand der Technik) als Ausgang des SCRR-ARB.

Dieses Signal wird gemäß der bekannten Tabelle dekodiert (**0803**) und auf ein Register zur Taktsynchronisierung (**0804**) geführt. Über dieses Register wird die DataIn Maske geschaltet. Dabei wird das Register entweder durch einen Takt oder ein Next-Signal (Enable EN), das den nächsten gültigen Binärvektor abfragt gesteuert. Bei einem Reset oder wenn die Kennung (ValidOut) ungültig anzeigt wird das Register so geschaltet, daß die DataIn Maske alle Signale durchschaltet.

Der Aufbau der Maske ist in **0805** dargestellt.

In **Fig.** 9 ist die LUT-Struktur abgebildet. Der Binärvektor (BinaryIn) des arbitrierten Triggers wird auf den Adresseingang der LUT1 (**0901**) geführt.

LUT1 übersetzt den Binärvektor entweder in einen gültigen Trigger um diesen an eine andere CT weiterzuleiten oder

eine gültige ID. Beide werden über **0910** ausgegeben. **0911** zeigt an, ob es sich um einen Trigger oder eine ID handelt. Ist über den Befehl "REFERENCE" keine Übersetzung des eingehenden Binärvektors in LUT1 eingetragen, wird – mittels eines Biteintrages oder eines Vergleichers auf ein bestimmtes Token (z. B. "VOID") – das Signal "Illegal Trigger" **0914** generiert.

Ein Trigger wird über **0912** an externe CTs geführt, IDs werden über den Multiplexer (**0902**) weiterverarbeitet. **0902** schaltet entweder der Datenausgang von LUT1, der eine gültige ID angibt, oder das ID-Register (**0903**) der CT auf den Adresseingang der LUT2 (**0904**). **0904** besitzt eine Cacheähnliche Struktur, d. h. der niederwertige Teil (**0906**) des Datenausgangs von **0902** wird auf den Adresseingang von **0904** geschaltet, während der höherwertige Teil (**0907**) auf den Dateneingang von **0904** geschaltet wird. Der **0907** gehörende Datenausgang wird über einen Komparator (**0905**) mit **0907** verglichen. Der Vorteil dieses Verfahrens ist, daß **0904** nicht die Tiefe zur Übersetzung aller IDs aufweisen muß, sondern erheblich kleiner ausfallen kann. Ähnlich eines gewöhnlichen Caches wird lediglich ein Teil der IDs übersetzt, wobei in der LUT2 anhand **0907** festgestellt werden kann, ob die selektierte ID der von LUT1 angegebenen entspricht. Dies entspricht einem Cache/TAG-Verfahren nach dem Stand der Technik.

Einem zweiten Dateneingang von **0904** ist ein Multiplexer **0908** zugeordnet, der je nach Operation den FreePointer (FP, Operation LOAD), den GarbagePointer (GP, Operation REMOVE) oder eine Tnvalid-Kennung/Token (NoAdr, Operation REMOVE) zur Speicherung an LUT2 liefert. Die beiden Pointer referenzieren auf Speicherstellen im CTR, "NoAdr" gibt an, daß kein Eintrag zu der passenden ID existiert, der Eintrag gelöscht wurde. Dies wird am Datenausgang festgestellt, indem über den Vergleicher **0909** die Daten auf das Token "NoAdr" verglichen werden.

An die Statemachine wird weitergeleitet:

- Das Auftreten eines Binärvektors wird über "ValidIn" (vgl. **Fig. 8**).
- Die Angabe ob es sich bei der Übersetzung in LUT1 um einen Trigger oder eine ID handelt (**0911**, "Trigger/ID Out"). Trigger werden über **0912** an andere CTs weitergeleitet, IDs werden in der eigenen CT abgearbeitet und an die LUT2 weitergeleitet.
- Das Ergebnis von **0905**, das angibt, ob die entsprechende ID in **0904** gespeichert ist ("Hit/Miss Out").
- Das Ergebnis von **0909**, das angibt, ob die entsprechende ID auf eine gültige Adresse im CTR zeigt ("NoEntry Out") Die von **0904** generierte Adresse wird an das CTR weitergeleitet ("CTR Address OutVV).

Die LUT1 wird über den Befehl "REFERENCE" mit der Übersetzung des eingehenden Binärvektors auf einen Trigger oder ID geladen. Die Operanden des Befehls werden über den Bus **0913** an die LUT1 geführt. Über denselben Bus wird das ID-Register (**0909**) geladen.

Fig. 10 zeigt die Pointerarithmetik des GarbagePointer (PG), ProgramPointer (PP), MovePointer (MP) und FreePointer (FP). Jeder Pointer besteht aus einem getrennt ansteuerbaren ladbaren up/down-Zähler. Jeder Zähler kann – sofern notwendig – mit dem Wert jedes anderen Zählers geladen werden; ebenso wie mit der Ausgabe von LUT2 (**1007**).

Über Vergleicher wird festgestellt ob

1. PP gleich MP
2. MP gleich FP
3. FP gleich der maximalen Position im CTR

ist. Die Ergebnisse werden zur Steuerung der Statemachines verwendet.

Über einen Multiplexer (**1001**) wird einer der Pointer zum Adresseingang des CTR geleitet. Die Daten gelangen über einen Multiplexer (**1002**) entweder von der übergeordneten CT (**1005**) oder aus einem Register (**1003**) an das CTR. Zur Statemachine und zum FILMO (**1006**) werden über einen Multiplexer (**1004**) entweder die Daten von der übergeordneten CT oder des CTR weitergeleitet. Dabei wird beim Auftreten eines REMOVE-Befehls von der übergeordneten CT der direkt über **1004** an die Statemachine geleitet, während ansonsten die Befehle aus dem CTR an die Statemachine geführt werden. Das Register **1003** dient zur Speicherung und Rückkopplung von Befehlen auf den CTR Eingang, die während eines Durchlaufs des Garbage-Kollektors von einer Adresse an eine andere geschoben werden.

Der Aufbau eines FILMOs ist in **Fig.** 11 dargestellt. Die Daten gelangen von dem CTR (**1101**) in das FILMO und werden entwerder über den Multiplexer (**1102**) in das FILMO-RAM (**1103**) geschrieben oder über den Multiplexer (**1104**) an die konfigruierbaren Elemente (**1116**) gesendet. Werden Daten in **1103** gelöscht, wird über **1102** eine "NOP"-Token nach **1103** geschrieben. Über den Vergleicher (**1105**) am Datenausgang wird das "NOP"-Token erkannt und ein Schreiben zu den konfigurierbaren Elementen verhindert. Über den Multiplexer **1106** wird entweder der Schreibzeiger WriteP (**1107**) oder der Lesezeiger (**1108**) an den Adresseingang von **1103** geführt. In dem Register **1109** wird der Lesezeiger gesichert um ein Rücksetzen (siehe **Fig.** 6) zu ermöglichen.

Der Füllstandszähler Full (**1110**) von **1103** wird gemäß **Fig.** 6 in dem Register **1111** zum Rücksetzen gespeichert. Zwei Vergleicher testen, ob **1103** leer (**1112**) oder voll (**1113**) ist. Über den Multiplexer **1115** wird selektiert, ob die Steuersignale der Statemachine (von **1101**) oder des FILMOs an **1116** gesendet wird.

Fig. 12a zeigt den hierarchischen Aufbau der CTs. Alle CTs beziehen ihre Daten aus der ROOT-CT (**1201**) und dem ihr zugeordneten ECR (**1204**). Für jede Implementierungsebene in einem Baustein existiert eine oder mehrere CTs. Jede CT ist für die Verwaltung ihrer Ebene und der darunterliegenden CTs zuständig. Es ist nicht notwendig, daß alle Äste das Baumes gleich tief sind. Beispielsweise können weniger Ebenen zur Steuerung der Peripherie (**1202**) eines Bausteines existieren als zur Steuerung der Arbeitseinheiten (**1203**) Der Datentransfer erfolgt baumartig. Jede CT arbeitet als Cache für alle unter ihr liegenden CTs.

Fig. 12b zeigt den Triggerfluß zwischen den CTs. Während der Datenfluß baumartig verläuft, ist der Triggerfluß nicht festgelegt. Jede CT kann an jede andere einen Trigger senden. Für gewöhnlich findet ein Triggeraustausch nur von den Blättern (**1203**) in Richtung der ROOT-CT (**1201**) statt. Unter Umständen kann der Transfer jedoch auch in die entgegengesetzte Richtung verlaufen.

In **Fig.** 12c ist ein Triggervektor Broadcast dargestellt, wobei **1205** einen Triggervektor an alle CTs sendet. **Fig.** 12d zeigt einen HIGHER-Triggervektor, den **1206** an die über ihr liegende CT sendet. **1207** sendet einen LOWER-Triggervektor an alle unter ihr liegenden CTs. **1208** überträgt einen direkt adressierten (ADDRESSED)-Triggervektor an eine bestimmte CT, die nicht direkt mit **1207** verbunden ist.

In **Fig.** 13 fordern zwei unabhängige IKR n und m eine gemeinsame, in der darüberliegenden CT gecachte KRx an. Es ist angedeutet, daß diese KR von dem gesamten Ast gecachet wird und auch in einem Nachbarast (**1301**) über eine gemeinsame CT verfügbar ist.

Fig. 14 zeigt ein gegenüber **Fig.** 9 modifiziertes LUT-System, das in ROOT-CTs und CTs mittlerer Hierarchieebenen verwendet wird. Der grundlegende Unterschied zu den bislang beschriebenen CTs ist, daß anstatt einzelner Triggersignale ID- und/oder Trigger-Vektoren von der CT verwaltet werden müssen. Jedem Vektor ist dabei ein Handshake-Signal (RDY, vgl. DE 197 04 728.9) zur Anzeige der Gültigkeit des Vektors zugeordnet, die an einen Arbiter (**1401**) geleitet werden. Über die Multiplexer (**1402, 1403**) wird entweder einer der Triggervektoren (**1404**) oder einer der ID-Vektoren (**1405**) ausgewählt. Triggervektoren gelangen direkt auf den Adresseingang der LUT1 (**1406**), die ansonsten gemäß **Fig.** 9 beschaltet ist. Das ID-Register (**1407**) ist ebenfalls gemäß **Fig.** 9 beschaltet. Im Gegensatz zu **Fig.** 9 besitzt der Multiplexer **1408** drei Eingänge (vgl. **0902**). Der Multiplexer wird dabei außer von der Statemachine zusätzlich von dem Arbiter **1401** angesteuert. Über den zusätzlichen Eingang werden ID-Vektoren über **1403** direkt an die LUT2 weitergeleitet. Dazu dient der Bus **1409**. "Trigger/ID Out" wird gemäß **Fig.** 9 generiert. Ein "ValidIn" Signal, das gemäß **Fig.** 9 auf ein "Valid Out" weitergeleitet wird existiert nicht. Statt dessen wird je nach Arbitrierung durch **1401** ein "Valid Trigger Out" für Triggervektoren und ein "Valid ID Out" für ID-Vektoren generiert, um die Statemachine anzuweisen, wie die Verarbeitung stattzufinden hat.

Der Bus **1409** wird über **1410** an eine weitere Einheit geleitet, die nur in der ROOT-CT existiert und in **Fig.** 15 beschrieben ist.

Eine ROOT-CT benötigt zusätzlich zu den normalen CT-Funktionen ein Interface zu dem externen Konfigurationsspeicher (ECR), sowie den erforderlichen Adressgenerator und Einheiten zum Verwalten der Zugriffe auf den ECR.

Eine gewöhnliche CT übersetzt in LUT1 eingehende Triggervektoren auf einen ID und in LUT2 das ID auf eine Speicherstelle im CTR (siehe **Fig.** 16a). Eine ROOT-CT übersetzt bei Zugriffen auf das ECR eine ID innerhalb des ECR auf eine Adresse im ECR, an der das durch TD referenziert KR/IKR beginnt. Dazu ist ein Speicherbereich im ECR festgelegt, dessen Größe der möglichen Anzahl an IDs entspricht (ist beispielsweise eine ID 10-bit breit, ergibt das $2^{10} = 1024$ mögliche IDs, also werden 1024 Einträge im ECR reserviert). In den folgenden Beispielen befindet sich dieser Speicherbereich am unteren Ende des ECRs und wird LUT-ECR genannt, um die Ähnlichkeit zur LUT2 zu unterstreichen. Die Übersetzung eines Triggers auf eine ID findet dabei gemäß den bereits bekannten CTs in der LUT1 statt (**1601**). Zum besseren Verständnis verdeutlicht **Fig.** 16b den Zugriff auf das ECR.

Eine ID gelangt in **Fig.** 15 über den Bus **1410** auf **Fig.** 14 an den Multiplexer **1501**. Über **1501** wird die ID in den ladbaren Zähler **1502** geschrieben. Der Ausgang von **1502** führt über einen Multiplexer **1503** an den Adressbus (**1504**) des ECR. Über den Datenbus **1505** gelangt die Übersetzung der ID auf eine Speicheradresse über einen Multiplexer/Demultiplexer (**1506**) an **1501**, der **1502** mit der Speicheradresse lädt. Daraufhin werden über die Statemachine LOAD-ECR (siehe **Fig.** 17) die Datenwörter der entsprechenden KR/IKR aus dem ECR gelesen und in das CTR geschrieben, wobei **1502** nach jedem Lesevorgang erhöht wird; so lange, bis der Befehl "STOP" gelesen wurde.

Über das Interface **1507** schreibt der übergeordnete HOST über **1503/1506** die KR/IKR in das ECR. Dabei wird über die Statemachine (CTS) arbitriert, ob der HOST oder die ROOT-CT Zugriff auf das ECR hat.

Nach einem Reset des Bausteines muß eine Grundkonfiguration (BOOT-KR) geladen werden. Dazu wird eine feste Speicheradresse (BOOT-ADR) eingeführt, die auf die erste Speicherstelle der BOOT-KR zeigt. Als BOOT-ADR wird die Speicherstelle 0h empfohlen, sofern die TDs bei 1 beginnen, andernfalls kann 2^{ID} oder irgend eine andere Speicherstelle verwendet werden. In dem Ausführungsbeispiel wird 2^{ID} verwendet.

Die ROOT-CT führt zum Laden der BOOT-KR an der Stelle BOOT-ADR einen Lookup durch, sofern eine BOOT-KR geladen ist. Die ROOT-CT schreibt die Daten nach **1502** um von dort die BOOT-KR bis zum Auftreten eines "STOP" Befehls zu laden.

Eine Überwachungseinheit innerhalb der ROOT-CT übernimmt die Synchronisation des HOST mit dem Baustein. Dies geschieht folgendermaßen:

Die Adressen kleine 2^{ID} werden durch **1508** überwacht, d. h. bei Zugriffen auf diese Adressen durch den HOST wird ein Signal (ACC-TD) an die Statemachine (CTS) gesendet. Ebenfalls wird BOOT-ADR über **1509** überwacht und sendet ein Signal ACC-BOOT an die Statemachine(CTS).

Die Statemachine (CTS) reagiert wie folgt:

- Schreibt HOST auf die BOOT-ADR, bewirkt dies das Laden der BOOT-KR.
- Schreibt HOST das Datenwort 0 (**1512**) auf die BOOT-ADR, wird dies über den Komparator **1510** festgestellt und bewirkt das Anhalten des Bausteines.
- schreibt der HOST auf eine Adresse kleiner 2^{ID} wird die Adresse in das REMOVE-Register (**1511**) geladen. Da die Adresse der ID entspricht (siehe ECR-LUT) steht die ID der geänderten KR/IKR in **1511**. An alle CTs wird der Befehl REMOVE <ID> zur sofortigen Ausführung gesendet (**1513**). Die CTs löschen daraufhin die KR/IKR der entsprechenden ID aus ihrem CTR, bzw. LUT2. Bei einem nachfolgenden Aufruf der KR/IKR müssen die CTs zwangsläufig die neue KR/IKR aus dem ECR laden.

Fig. 17 zeigt den Ablauf in einer ROOT-CT bei Laden einer KR/IKR aus dem ECR. Befindet sich eine ID nicht im internen CTR (vgl. **Fig.** 1, **1701**) wird die ID in den Zähler **1502** geschrieben (**1703**). Ein Zugriff auf das ECR mit der Adresse in **1502** liefert die Basisadresse der KR/IKR. Diese wird in **1502** geschrieben (**1704**). Ein LOAD gemäß **Fig.** 2 findet statt (**1702**). Dabei werden die Daten statt von einer Übergeordneten CT aus dem ECR gelesen (**1705**) und nicht nur in das eigene CTR geschrieben, sondern, an die untergeordnete CT gesendet (**1706**).

In einer CT mittlerer Hierarchieebene läuft die Übersetzung der Trigger ähnlich **Fig.** 1, mit der Ausnahme, daß Triggervektoren und ID-Vektoren gemäß **Fig.** 14 behandelt werden. Die KR/IKR werden gemäß **Fig.** 2 geladen, mit der Ausnahme, daß die Datenworte nicht nur in das eigene CTR geschrieben werden (**0210**), sondern gleichzeitig an die untergeordnete CT gesendet werden.

Fig. 19 verdeutlicht das FILMO Prinzip. Der FILMO (**1901**) wird bei lesenden und schreibenden Zugriffen immer vom Anfang zum Ende durchlaufen (**1902**). Werden Einträge vom Anfang des FILMOs geschrieben und gelöscht (**1903**), wird der Lesezeiger auf den ersten ungelöschten Eintrag verschoben (**1904**). Werden Einträge aus der Mitte das FTLMOs geschrieben (**1905**), bleibt der Lesezeiger unverändert (**1906**), die Einträge werden mit "NOP" markiert (**1907**) Werden Daten in das FILMO geschrieben (**1908**), werden diese am Ende, hinter dem letzten Eintrag angehängt (**1909**). Der Lesezeiger (**1910**) bleibt unverändert.

Selbstverständlich kann eine CT mit nur einem Speicher, der LUT1, LUT2 und CTR umfaßt aufgebaut werden. Die Steuerung dafür ist jedoch aufwendiger. Die CTs sind dabei ähnlich der ROOT-CT aufgebaut, die bereits die LUT2 UND das CTR im ECR integriert. Für das Verständnis des Verfahrens ist eine Beschreibung dieser CTs nicht erforderlich.

Wird eine CT als Cachesystem für Daten eingesetzt, werden Trigger zum Schreiben von Daten in das CTR eingeführt. Dabei werden die Daten von einer CEL in das CTR geschrieben. Die hierzu notwendigen Änderungen sind trivial, das FILMO kann komplett entfallen.

Beim Cachen der Daten tritt das Problem der Datenkonsistenz auf. Dies kann umgangen werden, indem ein Verfahren gemäß DE 42 21 278 A1 eingesetzt wird, um die Daten und deren Gültigkeit in den einzelnen Hierarchieebenen zu kennzeichnen. Werden Daten zur Durchführung eines Read-Modify-Write-Zyklusses (RMW-Zyklus) angefordert, werden die Daten auf allen Hierarchieebenen anhand eines zusätzlichen Eintrages in dem CTR/ECR als "ungültig" (INVALID) gekennzeichnet. In den Eintrag kann dazu die eindeutige ID der die Daten benutzenden KR/IKR eingetragen werden. Die Daten können so lange von keiner KR/IKR mit anderer ID benutzt werden, bis die die Daten benutzende KR/IKR die Daten zurückgeschrieben (vgl. Write-Back-Methode nach dem Stand der Technik) und ihre ID gelöscht hat.

Fig. 20 zeigt ein Ausführungsbeispiel:

In **Fig.** 20a fordert die CT **2007** Daten von der darüberliegenden CT an, diese fordert die Daten von der ROOT-CT **2004**; mit der Datenanforderung wird die ID der Anfordernden KR/IKR (**2001**) übertragen. Die Daten (**2002**) werden an **2007** gesendet. Alle anderen, späteren Zugriffe werden abgewiesen (**2003**).

In **Fig.** 20b werden die Daten zurückgeschrieben (**2005**), anderen, späteren Zugriffe werden wieder akzeptiert (**2006**) In **Fig.** 20c werden Daten von einer CT mittleren Hierarchie angefordert, im Besitz der Daten ist und diese an **2007** sendet. Die ID zum Sperren der Daten wird an alle CTs in der Hierarchie gesendet (**2001**). Beim Rückschreiben der Daten (Write-Back) in **Fig.** 20d werden die Daten an alle CTs in der Hierarchie geschrieben und die ID gelöscht.

Funktionskonvention

NICHT-Funktion!

I	Q
0	1
1	0

UND-Funktion &

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

ODER-Funktion #, ≥

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

TOR-Funktion G

EN	B	Q
0	0	-
0	1	-
1	0	0
1	1	1

Begriffsdefinition

Broadcast: Senden einer Information an eine Vielzahl von Empfängern.

Datenempfänger: Die Einheit(en), die Ergebnisse der CEL weiterverarbeitet/-arbeiten

Datensender: Die Einheit(en), die Daten für die CEL als Operanden zur Verfügung stellt/stellen

Datenwort: Ein Datenwort besteht aus einer beliebig langen Bit-Reihe. Diese Bit-Reihe stellt eine Verarbeitungseinheit für eine Anlage dar. In einem Datenwort können sowohl Befehle für Prozessoren o.ä. Bausteine sowie rein Daten kodiert werden.

Deadlock: Zustand, indem aufgrund gegenseitiger Blockade keinerlei Datenverarbeitung möglich ist.

DFP: Datenflußprozessor nach Patent/Offenlegung DE 44 16 881

DPGA: Dynamisch konfigurierbare FPGAs. Stand der Technik

Elemente: Sammelbegriff für alle Arten von in sich abgeschlossenen Einheiten, welche als Stück in einem elektronischen Baustein zum Einsatz kommen können. Elemente sind also:

- Konfigurierbare Zellen aller Art
- Cluster
- RAM-Blöcke
- Logik
- Rechenwerke
- Register
- Multiplexer
- I/O Pins eines Chips

Ereignis: Ein Ereignis kann durch ein Hardwareelement in irgendemer zur Anwendung passenden Art und Weise ausgewertet werden und als Reaktion auf diese Auswertung eine bedingte Aktion auslösen. Ereignisse sind somit zum Beispiel:

- Taktzyklus einer Rechenanlage.
- internes oder externes Interrupt-Signal.
- Trigger-Signal von anderen Elementen innerhalb des Bausteines.
- Vergleich eines Datenstroms und/oder eines Befehlstroms mit einem Wert.
- Input/Output Ereigenisse.
- Ablaufen, überlaufen, neusetzen etc. eines Zählers.
- Auswerten eines Vergleichs.

FIFO: First-In, First-Out Speicher nach dem Stand der Technik

FILMO: Abgewandeltes FIFO, aus dem linear Daten gelesen werden. Eine Beschränkung des Lesezeigers auf den Beginn des Speichers ist nicht vorhanden.

FPGA: Programmierbarer Logikbaustein. Stand der Technik.

F-PLUREG Register in dem die Funktion der CEL gesetzt wird. Ebenfalls wird der OneShot- und Sleep-Mode gesetzt. Das Register wird von der PLU beschrieben.

Fragmentierung: Zerteilen von Speicher in eine Vielzahl oftmals kleiner und damit nutzloser Speicherbereiche.

Garbage-Kollektor: Einheit zum Verwalten des Speichers. Verhindert eine Fragmentierung.

H-Pegel: Logisch 1 Pegel, abhängig von der verwendeten Technologie

HOST: Einem Baustein oder Baugruppe übergeordneter Rechner.

IDLE-Zyklus: Zyklus, in dem eine Statemachine keine Verarbeitung durchführt. Grundzustand einer Statemachine.

Pointer: Zeiger auf eine Adresse bzw. ein Datenwort.

konfigurierbares Element: Ein konfigurierbares Element stellt eine Einheit eines Logik-Bausteines dar, welche durch ein Konfigurationswort für eine spezielle Funktion eingestellt werden kann. Konfigurierbare Elemente sind somit, alle Arten von RAM Zellen, Multiplexer, Arithmetische logische Einheiten, Register und alle Arten von interner und externer Vernetzungsbeschreibung etc.

konfigurierbare Zelle: Siehe Logikzellen

Konfigurieren: Einstellen der Funktion und Vernetzung einer logischen Einheit, einer (FPGA)-Zelle oder einer CEL (vgl. umkonfigurieren).

Konfigurationsdaten: Beliebige Menge von Konfigurationsworten.

Konfigurationsroutine: Mehrere Konfigurationsworte zu einem Algorithmus zusammengefügte.

Konfigurationsspeicher: Der Konfigurationspeicher enthält ein oder mehrere Konfigurationsworte.

Konfigurationswort: Ein Konfigurationswort besteht aus einer beliebig langen Bit-Reihe. Diese Bit-Reihe stellt eine gültige Einstellung für das zu konfigurierende Element dar, so das eine funktionsfähige Einheit entsteht.

Ladelogik: Einheit zum Konfigurieren und Umkonfigurieren der CEL. Ausgestaltet durch einen speziell an seine Aufgabe angepaßten Mikrokontroller.

Logikzellen: Bei DFPs, FPGAs, DPGAs verwendete konfigurierbare Zellen, die einfache logische oder arithmetische Aufgaben gemäß ihrer Konfiguration erfüllen.

Lookup-Tabelle: Stand der Technik. Verfahren zum Übersetzen von Daten.

L-Pegel: Logisch 0 Pegel, abhängig von der verwendeten Technologie.

Maske: Bitkombination, die die gültigen Signale innerhalb einer Mehrzahl von Signalen angibt.

PLU: Einheit zum Konfigurieren und Umkonfigurieren der CEL. Ausgestaltet durch einen speziell an seine Aufgabe angepaßten Mikrokontroller.

Priorisierung: Festlegung einer Reihenfolge.

PECONFIG: Rekonfigurierbarer Zustand einer CEL.

RECONFIG-Trigger: Setzen einer CEL in den rekonfigurierbaren Zustand.

REMOVE-<ID>: 1. Befehl innerhalb eines KR zum Entfernen der durch ID referenzierten KR. 2. Befehl einer übergeordneten CT über ein separates Interface oder Handshaking an eine untergeordnete CT zum löschen der durch ID referenzierten KR.

RESET: Rücksetzen eines Bausteines oder eines ganzen Computersystems in einen definierten Grundzustand.

ROOT-CT: CT der höchsten Hierarchieebene mit direktem Zugriff auf den externen Konfigurationsspeicher.

Round-Pobin-Arbiter: Arbiter der im Kreis läuft und immer dem zuletzt arbitrierten Signal die niederste Priorität zuordnet.

Statemachine: siehe Zustandsmaschine.

Switching-Tabelle: Eine Switching-Tabelle ist ein Ring-Speicher, welcher durch eine Steuerung angesprochen wird. Die Einträge einer Switching-Tabelle können beliebige Konfigurationswörter aufnehmen. Die Steuerung kann Befehle durchführen. Die Switching-Tabelle reagiert auf Triggersignale und konfiguriert konfigurierbare Elemente anhand eines Eintrages in einem Ringspeicher um.

Synchronisationssignale: Statussignale die von einem konfigurierbaren Element oder einem Rechenwerk generiert werden und zur Steuerung und Synchronisation der Datenverarbeitung an weitere konfigurierbare Element oder Rechenwerke weitergeleitet werden. Es ist auch möglich ein Synchronisationssignal zeitlich verzögert (gespeichert) an ein und dasselbe konfigurierbare Element oder Rechenwerk zurückzuleiten.

Terminierung: Beendigung eines Vorganges, einer Verarbeitung.

TRIGACK/TRIGRDY: Handshake der Trigger. (Vgl. Handshake der Daten aus P19651075.9)

Trigger: Synonym für Synchronisationssignale.

Umkonfigurieren: Neues Konfigurieren von einer beliebigen Menge von CELs während eine beliebige Restmenge von CELs ihre eigenen Funktionen fortsetzen (vgl. konfigurieren).

Verarbeitungszyklus: Ein Verarbeitungszyklus beschreibt die Dauer, welche von einer Einheit benötigt wird, um von einem definierten und/oder gültigen Zustand in den nächsten definierten und/oder gültigen Zustand, zu gelangen.

Zellen: Synonym für konfigurierbare Elemente.

Zustandsmaschine: Logik, die diversen Zuständen annehmen kann. Die Übergänge zwischen den Zuständen sind von verschiedenen Eingangsparametern abhängig. Diese Maschinen werden zur Steuerung komplexer Funktionen eingesetzt und entsprechen dem Stand der Technik.

Patentansprüche

1. Verfahren zum deadlockfreien, automatischen Konfigurieren und Rekonfigurieren von Bausteinen mit zwei- oder mehrdimensionaler Zellanordnung (z. B. FPGAs, DPGAs, DFPs gemäß DE 44 16 881 A1, o. dgl.), **dadurch gekennzeichnet**, daß eine Einheit zur Steuerung der Konfiguration und Rekonfiguration eine Menge von zugeordneten konfigurierbaren Elementen verwaltet, wobei die Menge eine Teilmenge aller konfigurierbaren Elemente oder die Gesamtmenge aller konfigurierbaren Elemente ist, und die Verwaltung wie folgt abläuft

   1.1 Rekonfigurationsanforderungen von den zugeordneten konfigurierbaren Elementen an die Einheit gesendet werden

   1.2 die Einheit die Anforderungen bearbeitet, indem

      a) keine weiteren Anforderungen und Veränderungen während der Ausführung akzeptiert werden,

b) die noch zu ladenden Konfigurationsdaten bestehender früherer Anforderungen aus einem Zwischenspeicher (FILMO) in die konfigurierbaren Elemente geladen werden,

c) die bestehende Anforderung auf eine eindeutige Kennung (ID) umgesetzt wird

1.3 die ID auf die Adresse der zu ladenden Konfigurationsdaten im Speicher der Einheit umgesetzt wird, sofern die Konfigurationsdaten im Speicher der Einheit existieren,

1.4 die Konfigurationsdaten mit der entsprechenden ID bei einer übergeordneten Einheit angefordert und geladen werden, sofern die Konfigurationsdaten nicht im Speicher der Einheit existieren,

1.5 die Konfigurationsdaten in die konfigurierbaren Elemente geladen werden, sofern die konfigurierbaren Elemente die Daten annehmen können,

1.6 die Konfigurationsdaten der konfigurierbaren Elemente, die die Daten nicht annehmen können in einen Zwischenspeicher geladen werden,

1.7 nachdem die Konfigurationsdaten vollständig abgearbeitet sind, wieder neu auftretende Anforderungen akzeptiert werden, wobei bis zum Auftreten einer erneuten Anforderung die noch zu ladenden Konfigurationsdaten bestehender früherer Anforderungen aus einem Zwischenspeicher (FILMO) in die konfigurierbaren Elemente geladen werden.

2. Verfahren nach Anspruch 1, dadurch gekennzeichnet, daß Konfigurationsdaten, die nicht an die konfigurierbaren Elementen angenommen worden sind, an das Ende des Zwischenspeichers geschrieben werden.

3. Verfahren nach Anspruch 1 und 2, dadurch gekennzeichnet, daß Konfigurationsdaten zum Schreiben in die konfigurierbaren Elemente vom Anfang zum Ende des Zwischenspeichers gelesen werden, wobei der Zwischenspeicher immer komplett durchlaufen wird.

4. Verfahren nach Anspruch 1 bis 3, dadurch gekennzeichnet, daß Konfigurationsdaten, die am Anfang des Zwischenspeichers in die konfigurierbaren Elemente geschrieben werden, im Zwischenspeicher gelöscht werden, d. h. der Lesezeiger wird hinter die geschriebenen Daten gesetzt.

5. Verfahren nach Anspruch 1 bis 4, dadurch gekennzeichnet, daß Konfigurationsdaten, die aus der Mitte des Zwischenspeichers in die konfigurierbaren Elemente geschrieben werden,

a) entweder im Zwischenspeicher als gelöscht markiert werden und damit bei einem erneuten Lesedurchlauf übersprungen werden,

b) oder im Zwischenspeicher (FILMO) gelöscht werden, indem die noch existierenden Konfigurationsdaten ausgehend von der zu löschenden Speicherstelle bis zum Anfang oder zum Ende des Zwischenspeichers (FILMOs) so verschoben werden, daß die gelöschte Speicherstelle mit bestehenden Konfigurationsdaten belegt ist und die Zeiger auf den Beginn oder das Ende des Speichers entsprechend angepaßt werden.

6. Verfahren nach Anspruch 1 bis 5, dadurch gekennzeichnet, daß eine Zustandsmaschine (Garbage-Kollektor) den Speicher (CTR) der Einheit so verwaltet, daß keine Speicherlücken entstehen, indem Speicherlücken mit existierenden Konfigurationsdaten ausgehend vom Beginn der Speicherlücke bis zum Ende des Speichers (CTR) so verschoben werden, daß die gelöschte Speicherstelle mit bestehenden Konfigurationsdaten belegt ist und die Zeiger auf das Ende des Speichers und die Übersetzung der IDs auf verschobene Speicherstellen entsprechend angepaßt werden.

7. Verfahren nach Anspruch 1 bis 6, dadurch gekennzeichnet, daß bei der Einheit eingehende Anforderungen auf eine eindeutige Kennung (ID) einer Konfigurationsroutine übersetzt wird, und die ID auf eine Speicherstelle im Speicher (CTR) zeigt.

8. Verfahren nach Anspruch 1 bis 7, dadurch gekennzeichnet, daß Konfigurationsdaten das Laden weiterer Konfigurationsdaten bewirken (EXECUTE).

9. Verfahren nach Anspruch 1 bis 8, dadurch gekennzeichnet, daß Konfigurationsdaten mittels Anforderung an eine andere Einheit das Ausführen weiterer Konfigurationsdaten in der anderen Einheiten bewirken (TRIGGER), wobei die Anforderungen

a) als Broadcast an alle anderen Einheiten gesendet werden,

b) nur an die direkt übergeordnete Einheit gesendet werden,

c) nur an die direkt untergeordnet(n) Einheit(en) gesendet werden

d) an eine bestimmte, adressierte Einheit gesendet werden.

10. Verfahren nach Anspruch 1 bis 9, dadurch gekennzeichnet, daß die Einheiten hierarchisch in einer Baumstruktur angeordnet sind.

11. Verfahren nach Anspruch 1 bis 10, dadurch gekennzeichnet, daß die höchste Einheit in der Baumstruktur einen gemeinsamen Speicher mit einem übergeordneten Rechner teilt und der übergeordnete Rechner die Abläufe in dem Baustein über Kommunikation mit der übergeordneten Einheit steuert.

12. Verfahren nach Anspruch 1 bis 11, dadurch gekennzeichnet, daß Einheiten mittlerer und höchster Hierarchieebene neben gewöhnlichen Anforderungen (Trigger) auch auf Anforderungen von IDs reagieren, wobei die Übersetzung der ID in die Adresse der Speicherstelle der referenzierten Konfigurationsroutine erfolgt und die Übersetzung eines Triggers in eine ID entfällt.

13. Verfahren zum Cachen von Daten und Befehlen in aus mehreren Rechenwerken bestehenden Mikroprozessoren und in Bausteinen mit zwei- oder mehrdimensionaler Zellanordnung (z. B. FPGAs, DPGAs, DFPs gemäß DE 44 16 881 A1, o. dgl.), dadurch gekennzeichnet, daß

13.1 mehrere Zellen und Rechenwerke (CEL) zu einer Mehrzahl von Gruppen zusammengefaßt werden, wobei jeder Teilgruppe ein Cache (CT) zugeordnet ist,

13.2 die Caches der einzelnen Teilgruppen werden über eine Baumstruktur an einen übergeordneten Cache (ROOT-CT) geschaltet, der Zugriff auf einen Speicher (ECR) besitzt, in dem die Daten/Befehle abgelegt sind,

13.3 jeder untergeordnete Cache oder Cache auf mittlerer Ebene des Baumes fordert die benötigten Daten/Befehle bei dem jeweils übergeordneten Cache an,

13.4 die Daten werden über einen Write-Through-Mechanismus an die übergeordneten CTs und in den Speicher (ECR) geschrieben werden,

13.5 die Daten werden während des Zugriffes durch eine KR/IKR in allen CTs durch Eintragen der ID der betreffenden KR/TKR in das CTR/ECR als "ungültig" markiert,

13.6 beim Rückschreiben der Daten wird die ID in dem zusätzlichen Eintrag des CTR/ECR gelöscht, wodurch die Daten als gültig markiert sind.

14. Verfahren zum Arbitrieren von Daten dadurch gekennzeichnet, daß

14.1 die zu arbitrierenden Signale über eine Maske auf einen Round-Robin-Arbiter geleitet werden, wobei der Grundzustand der Maske alle Signale durchschaltet,

14.2 der Arbiterausgang gibt das arbitrierte Signal an,

14.3 das arbitrierte Signal wird so dekodiert und auf die Maske zurückgeführt, daß das Signal und alle anderen höherpriorisierten Signale von der Maske gesperrt werden,

14.4 ist das letzte Signal arbitriert oder es existiert kein von der Maske freigeschaltetes Signal, das eine Arbitrierung anfordert, wird die Maske komplett freigeschaltet.

Hierzu 21 Seite(n) Zeichnungen

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

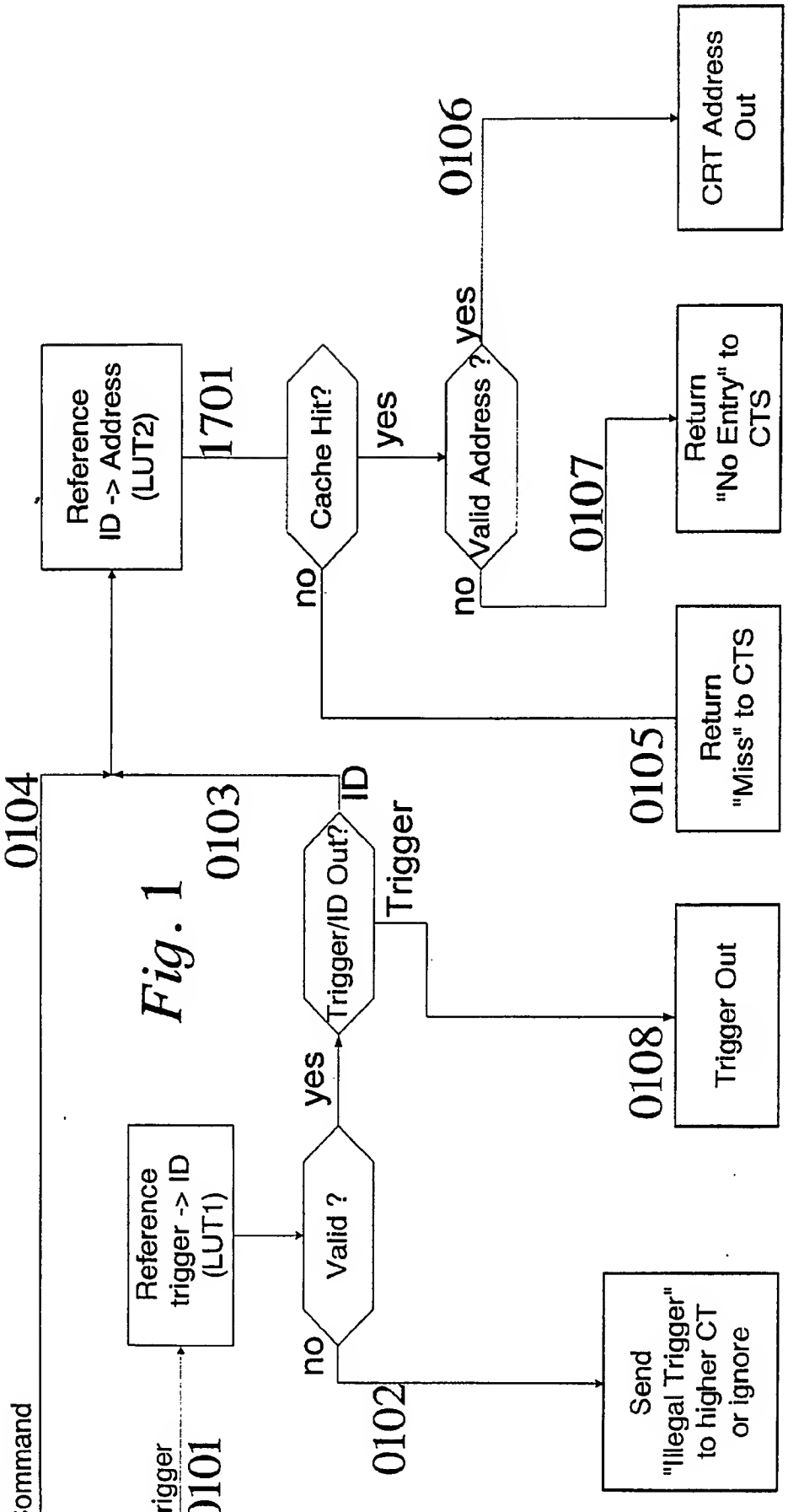
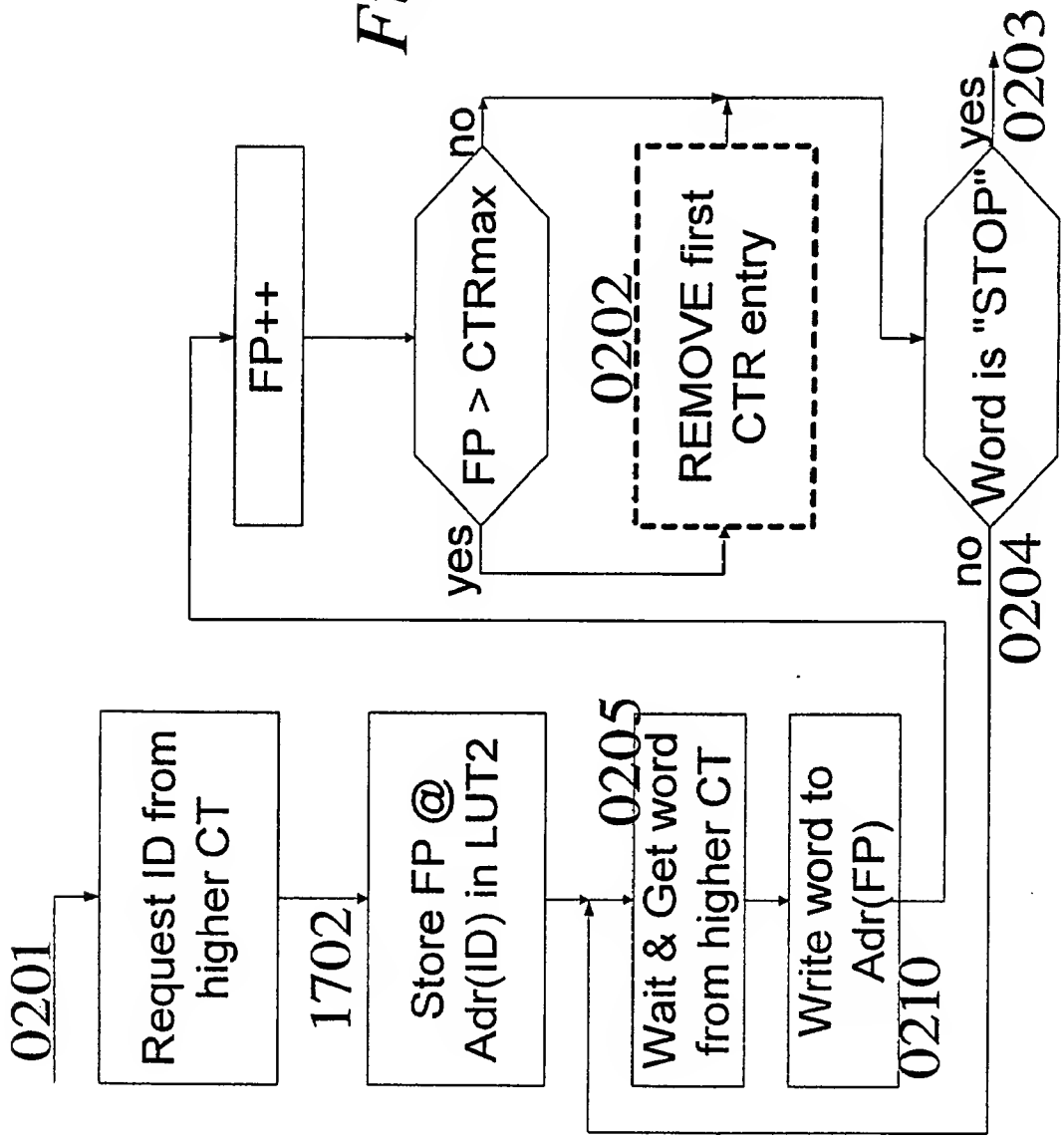


Fig. 1

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

Fig. 2



Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

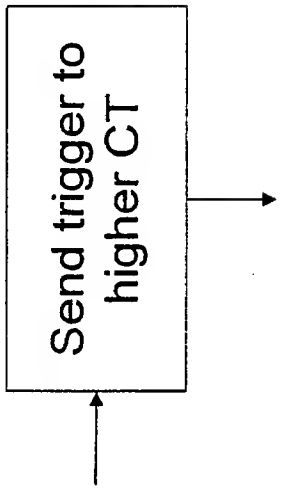


Fig. 3b

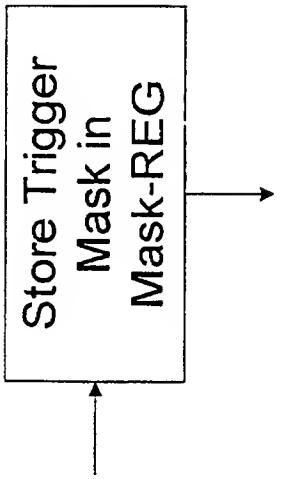


Fig. 3a

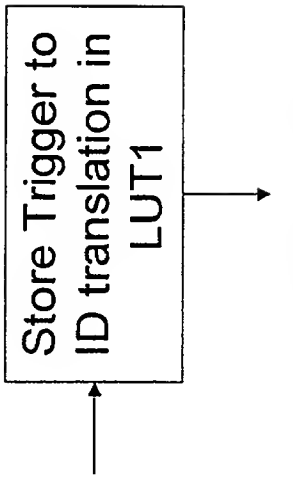


Fig. 3c

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

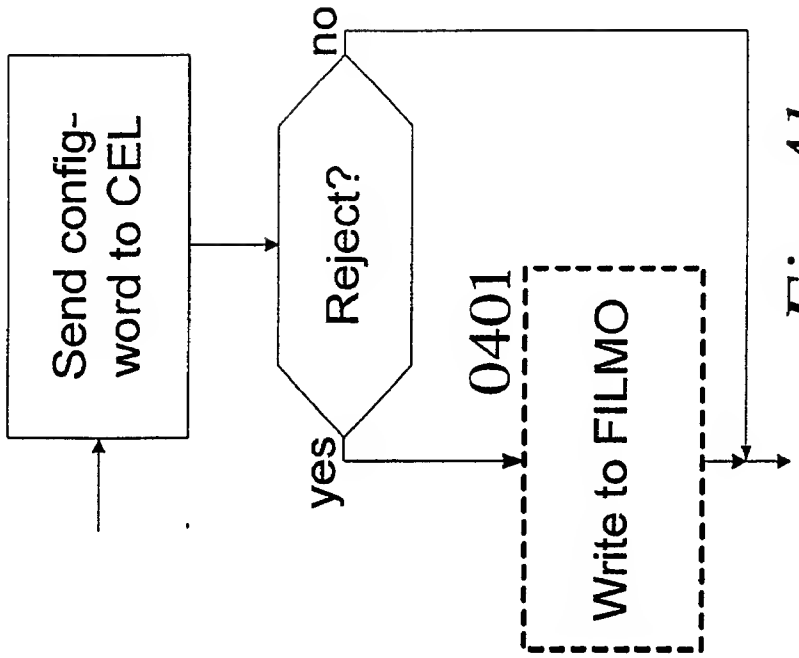


Fig. 4b

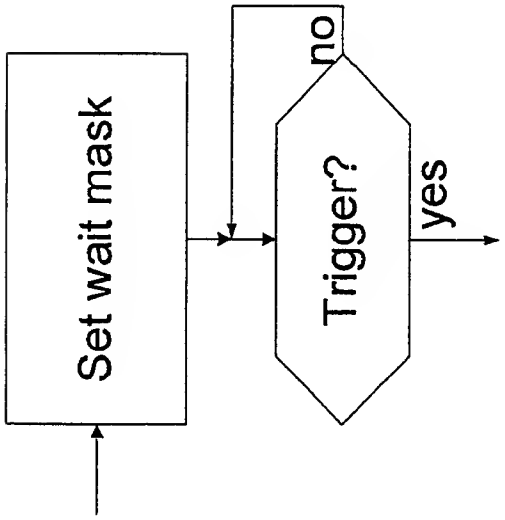
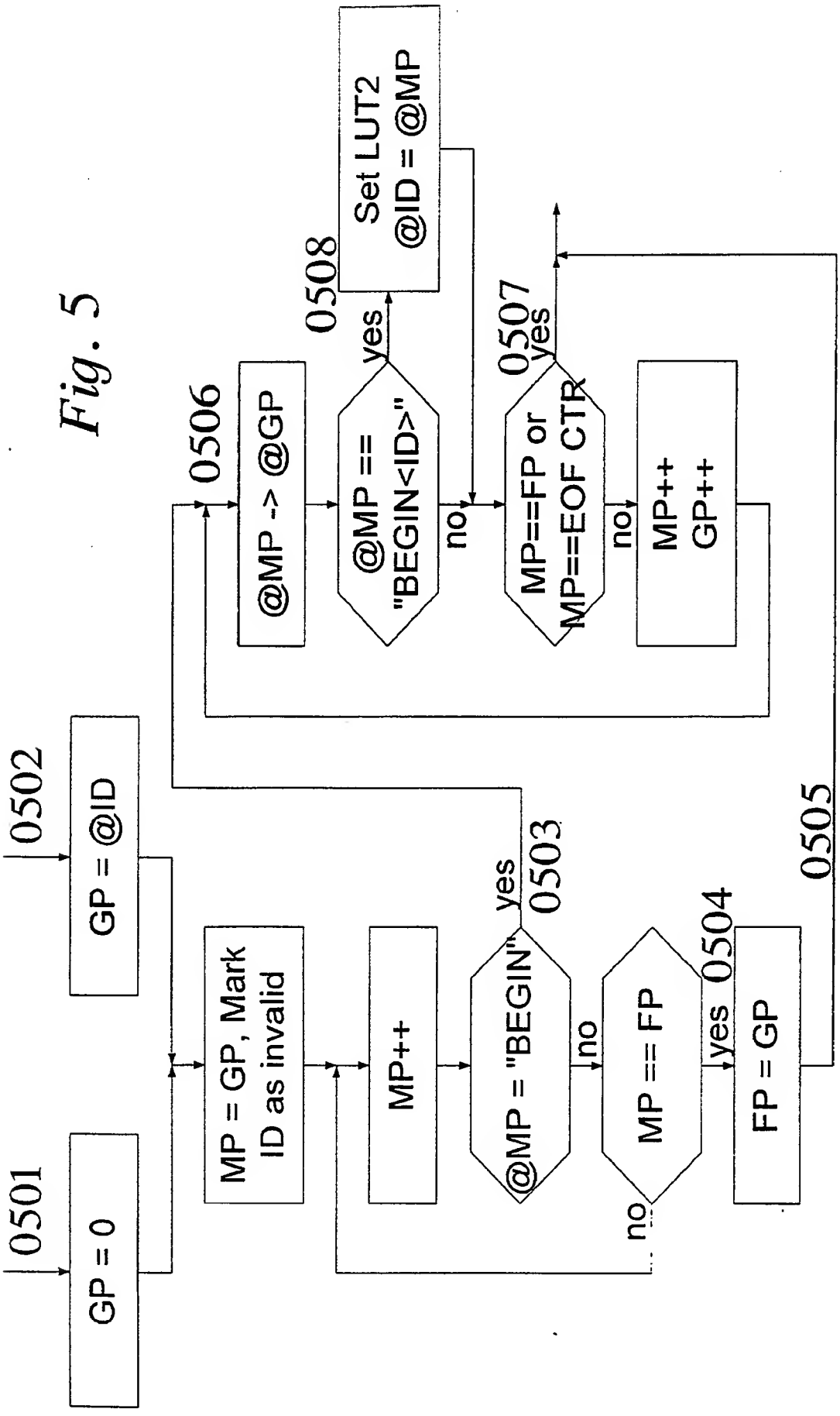


Fig. 4a

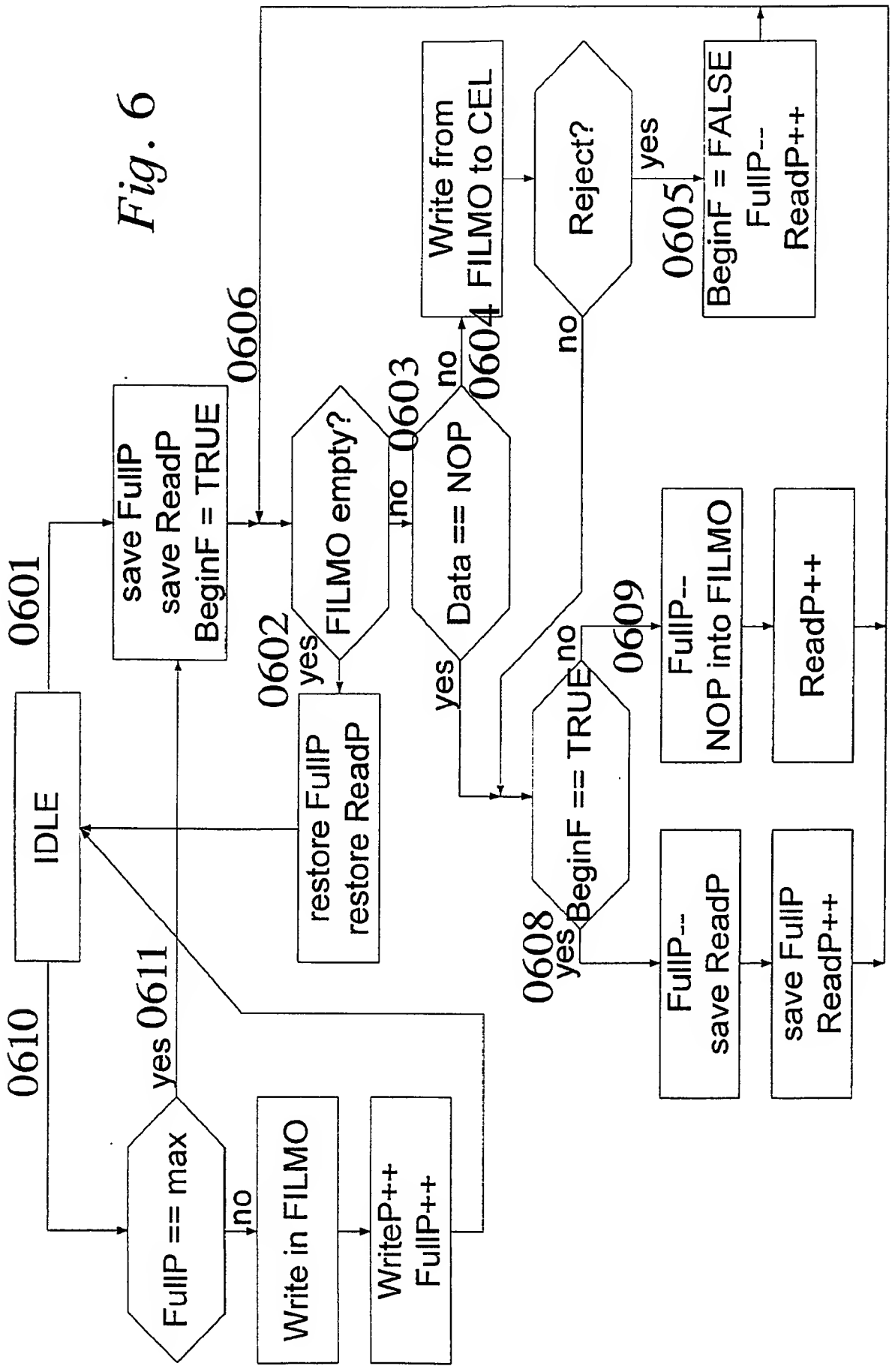
Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

Fig. 5



Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

Fig. 6



Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

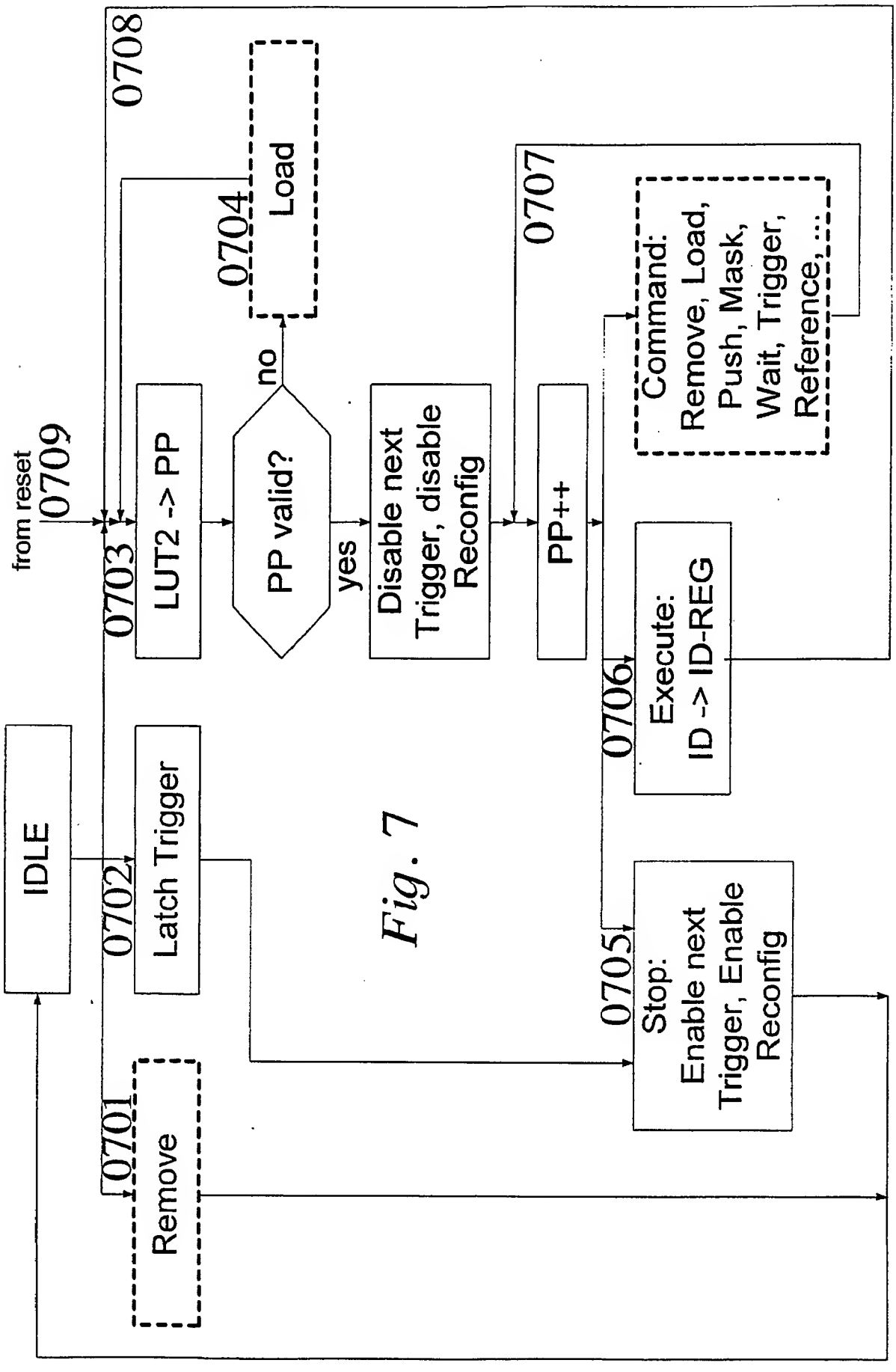


Fig. 7

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

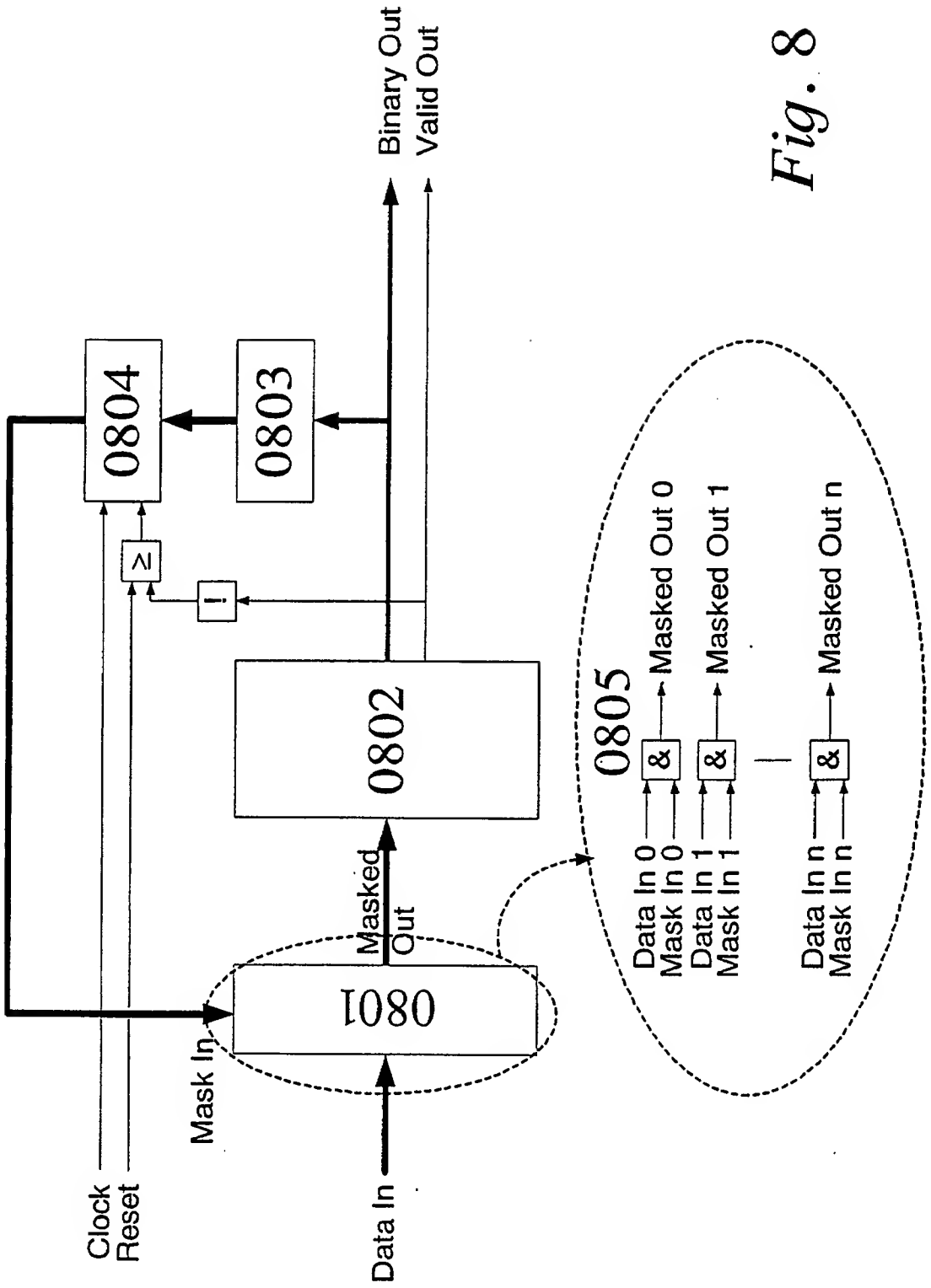


Fig. 8

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

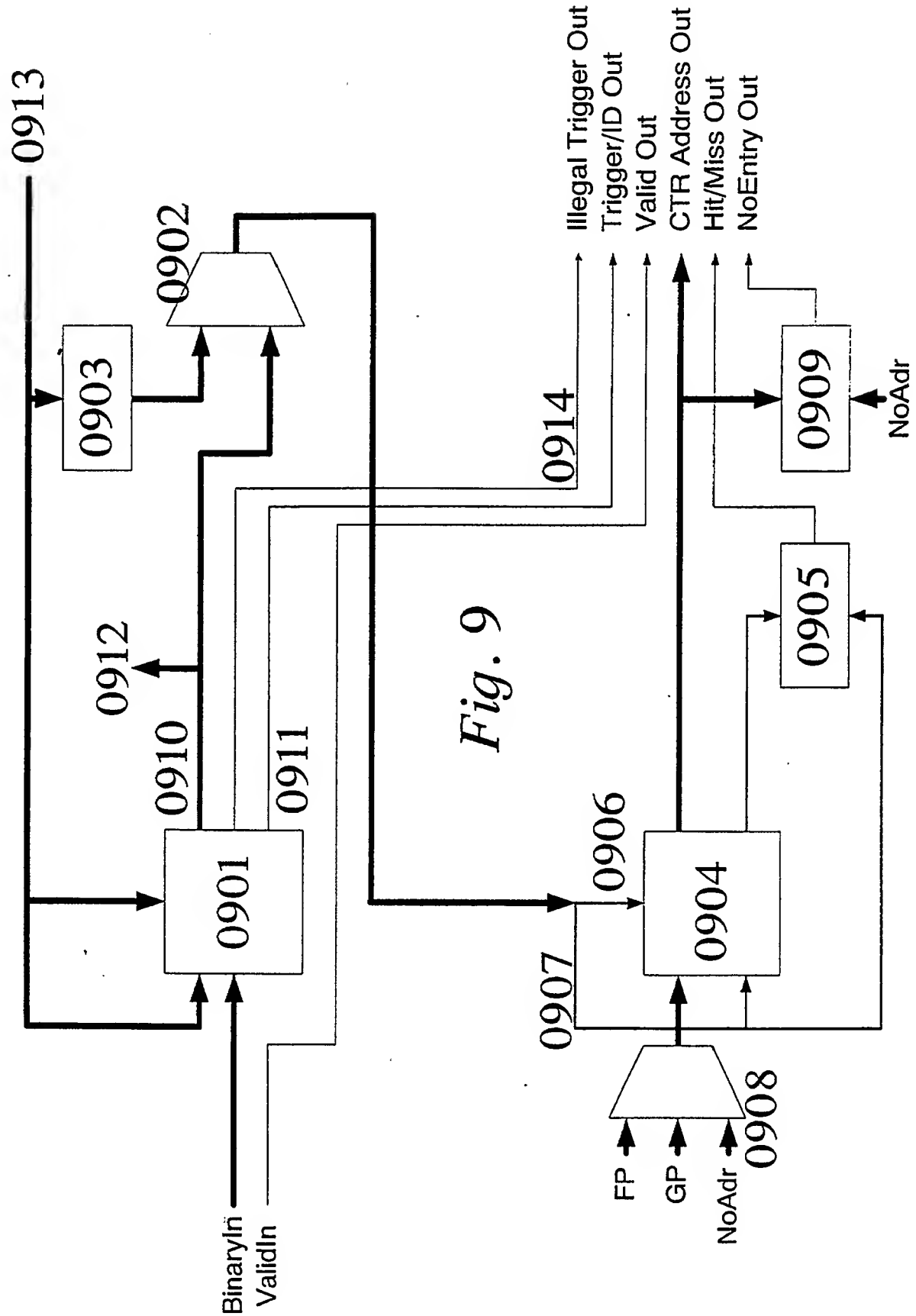


Fig. 9

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

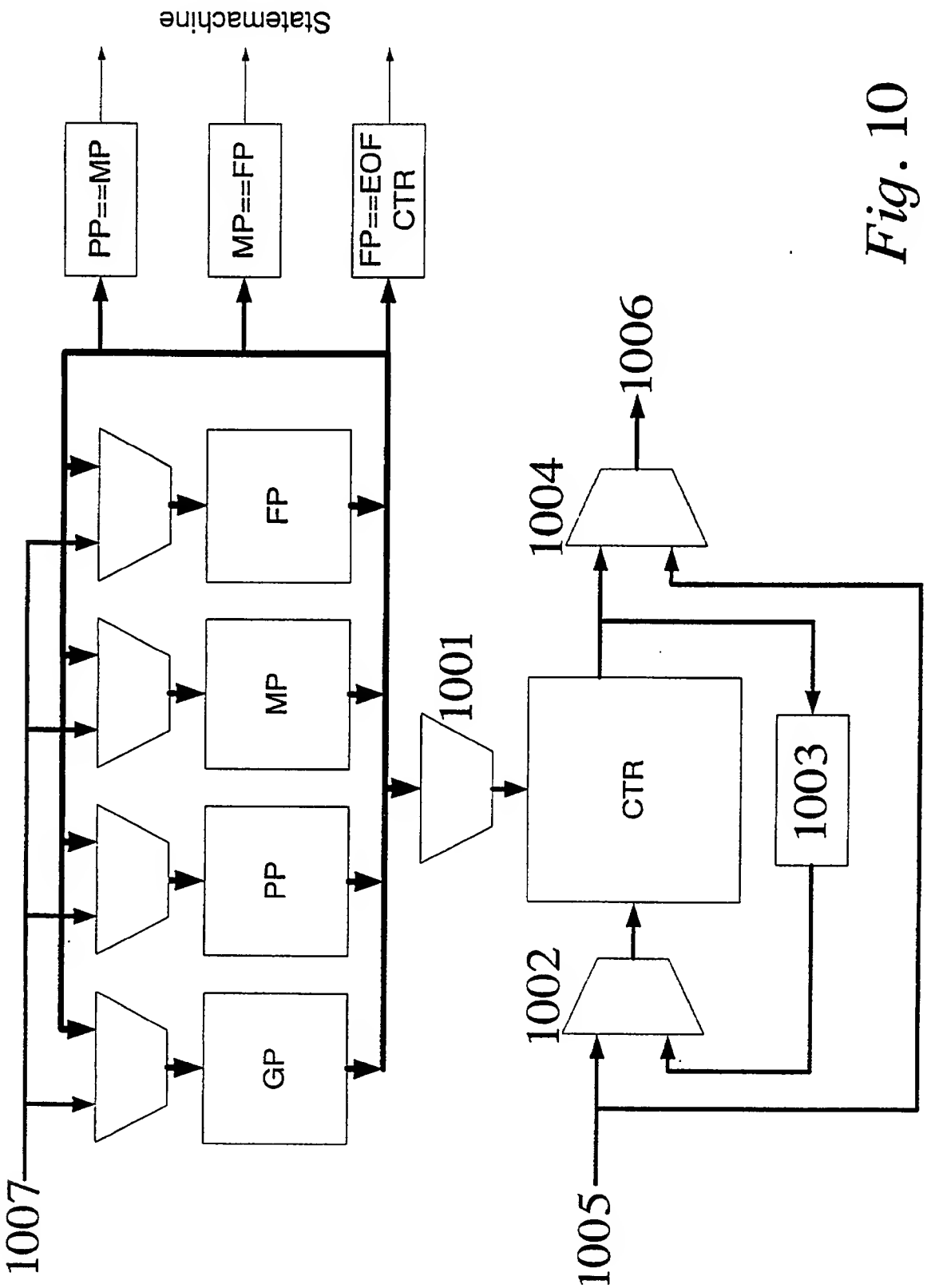


Fig. 10

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

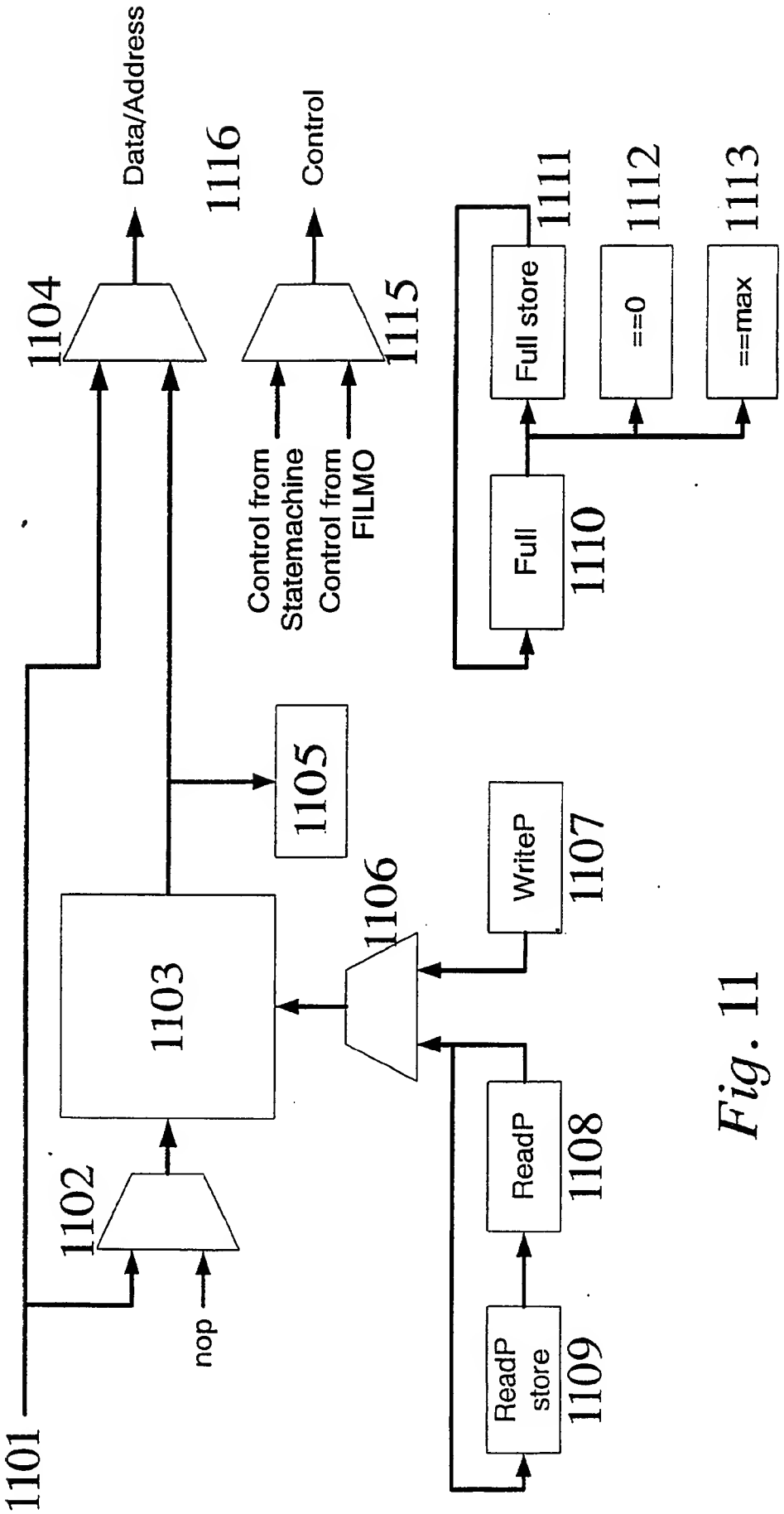


Fig. 11

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

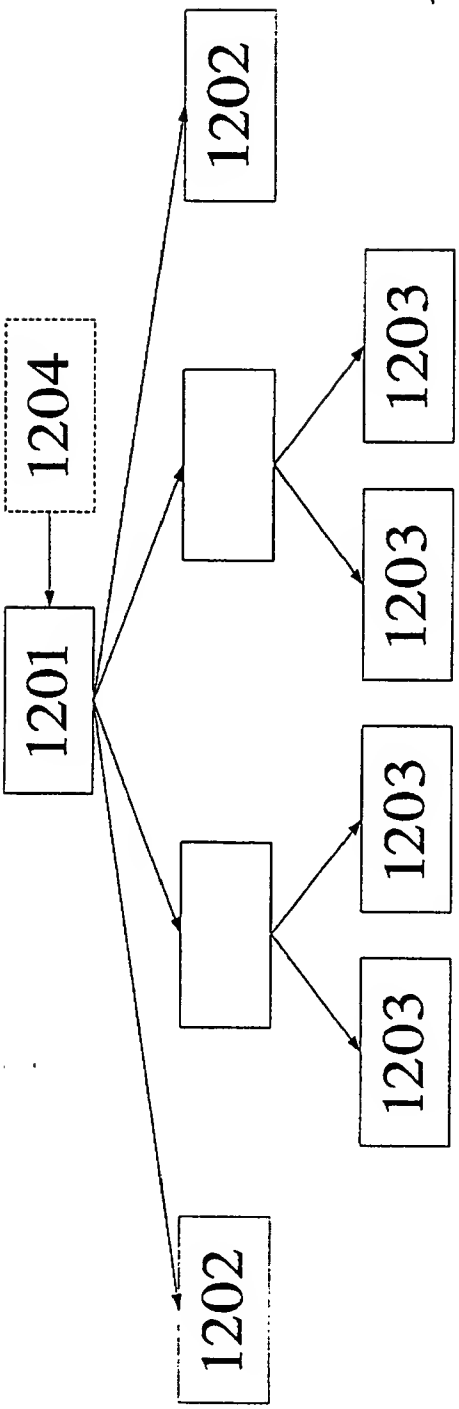


Fig. 12a

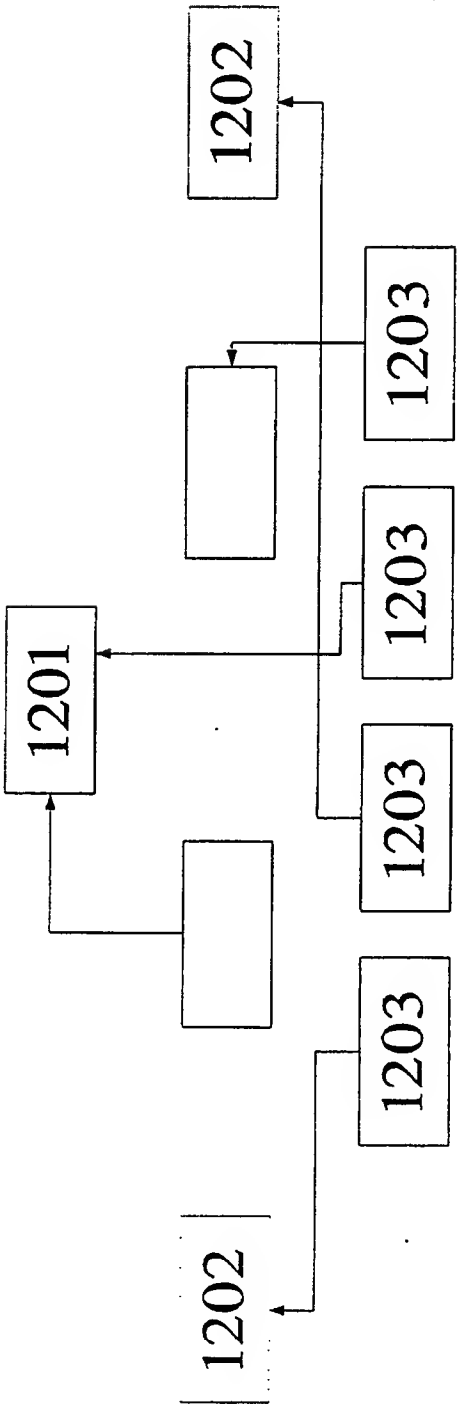


Fig. 12b

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

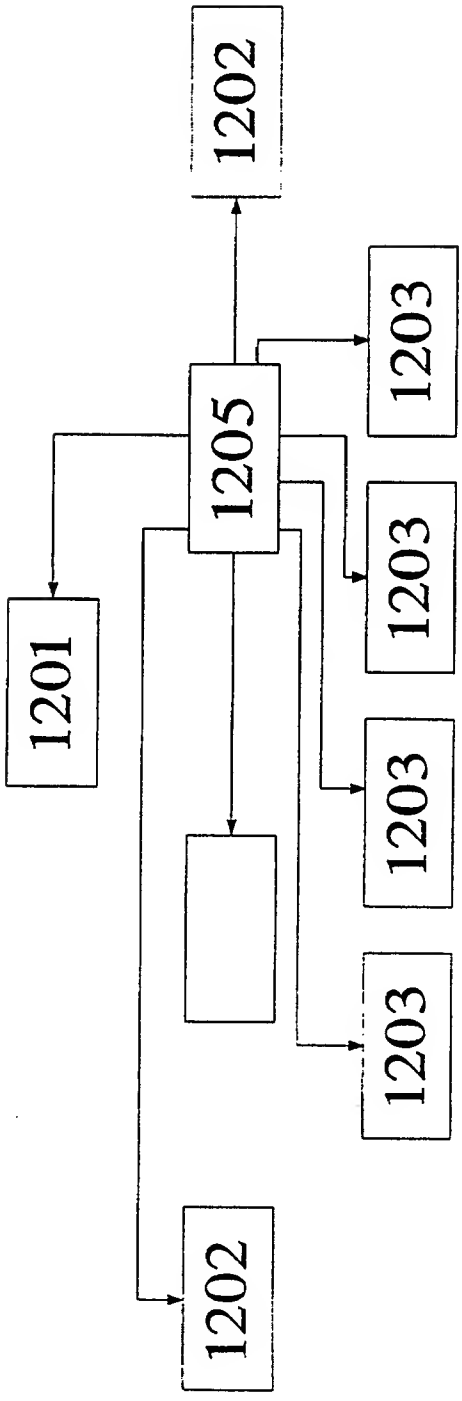


Fig. 12c

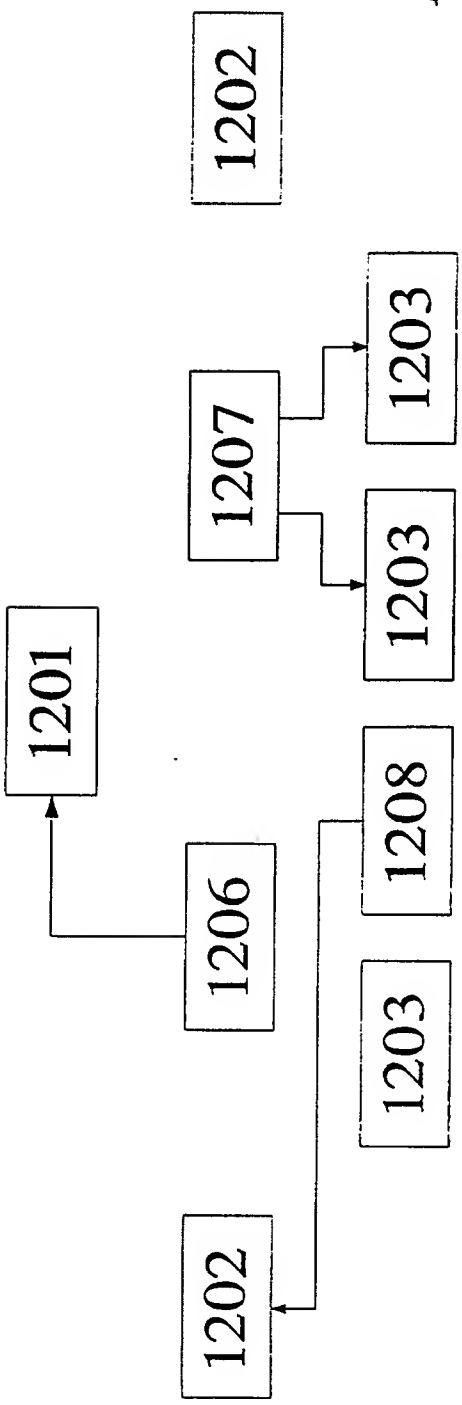


Fig. 12d

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

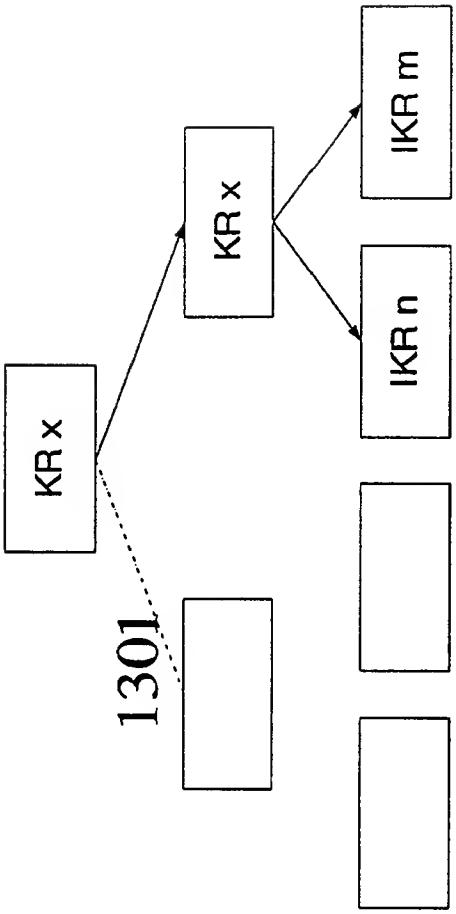


Fig. 13

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

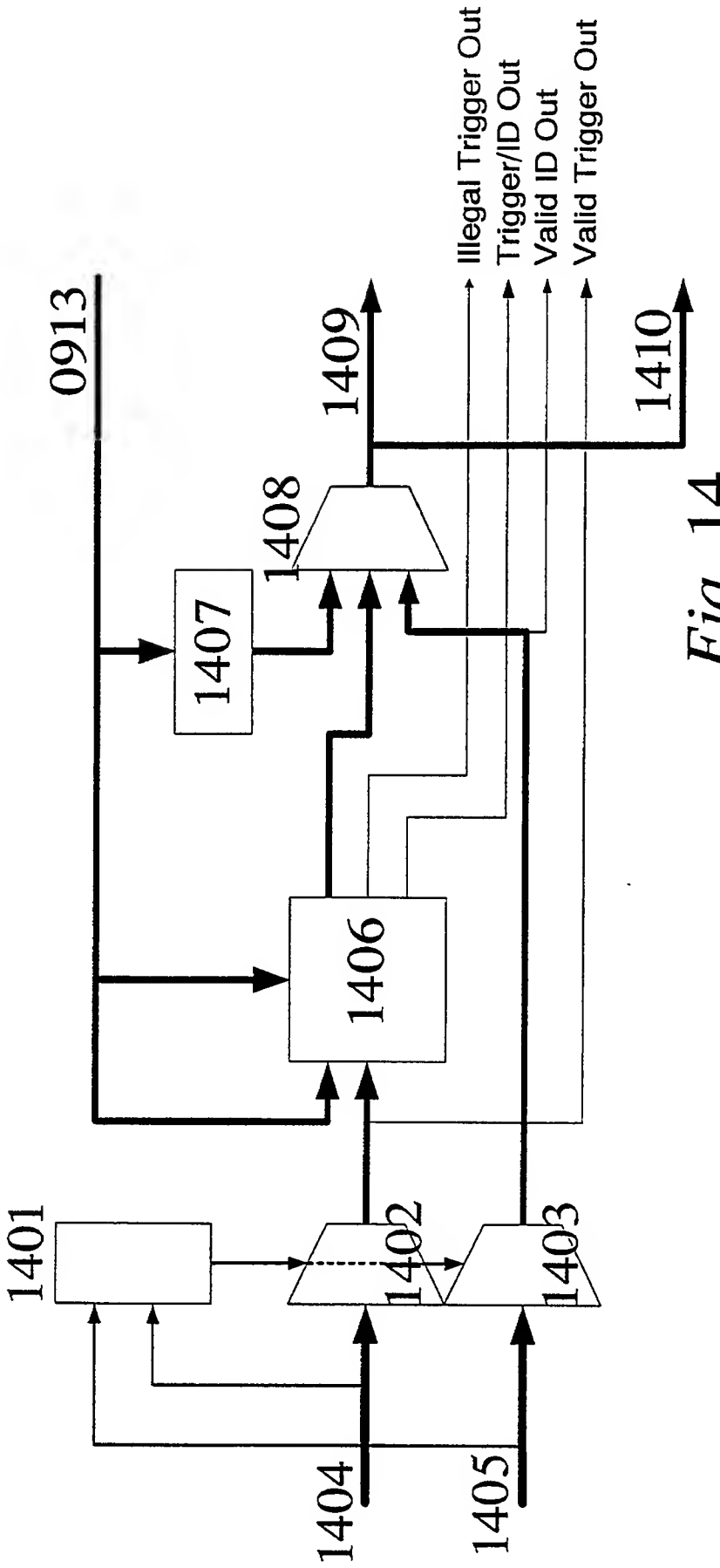


Fig. 14

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

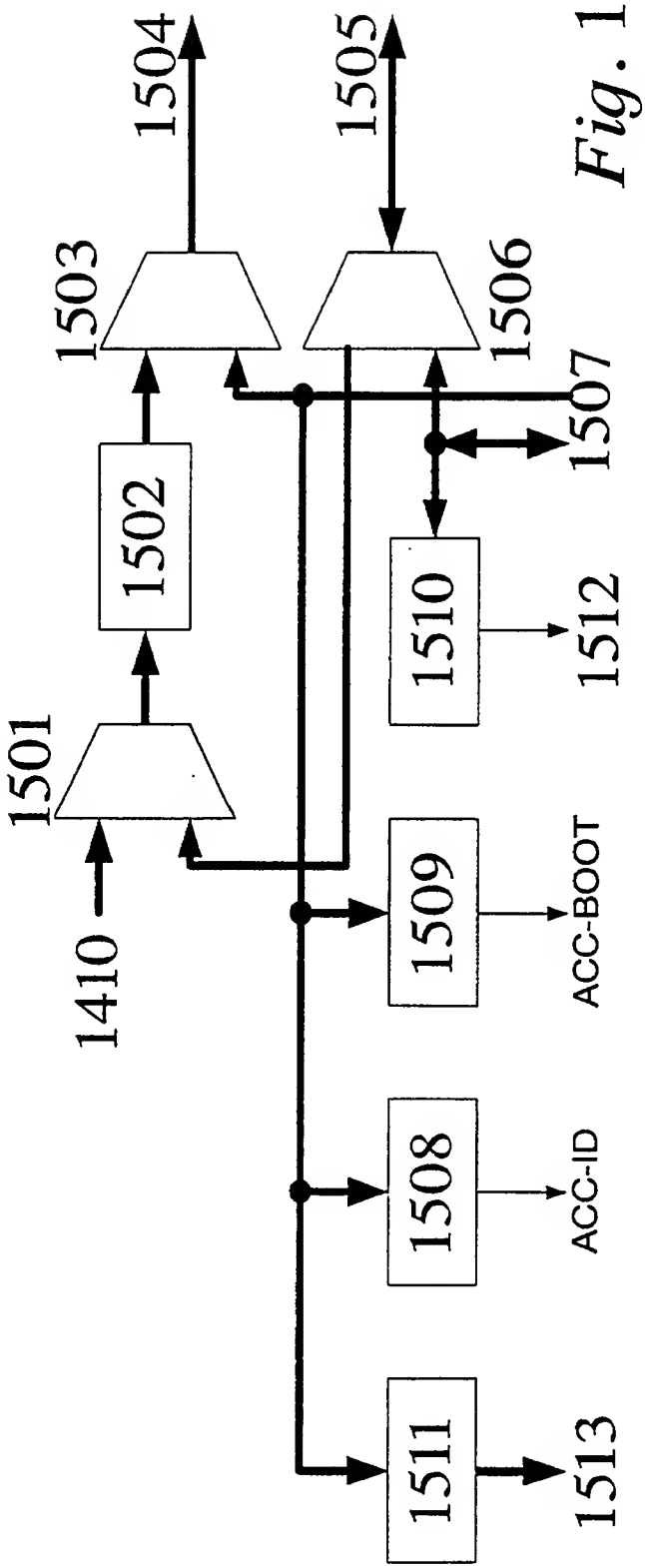


Fig. 15

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

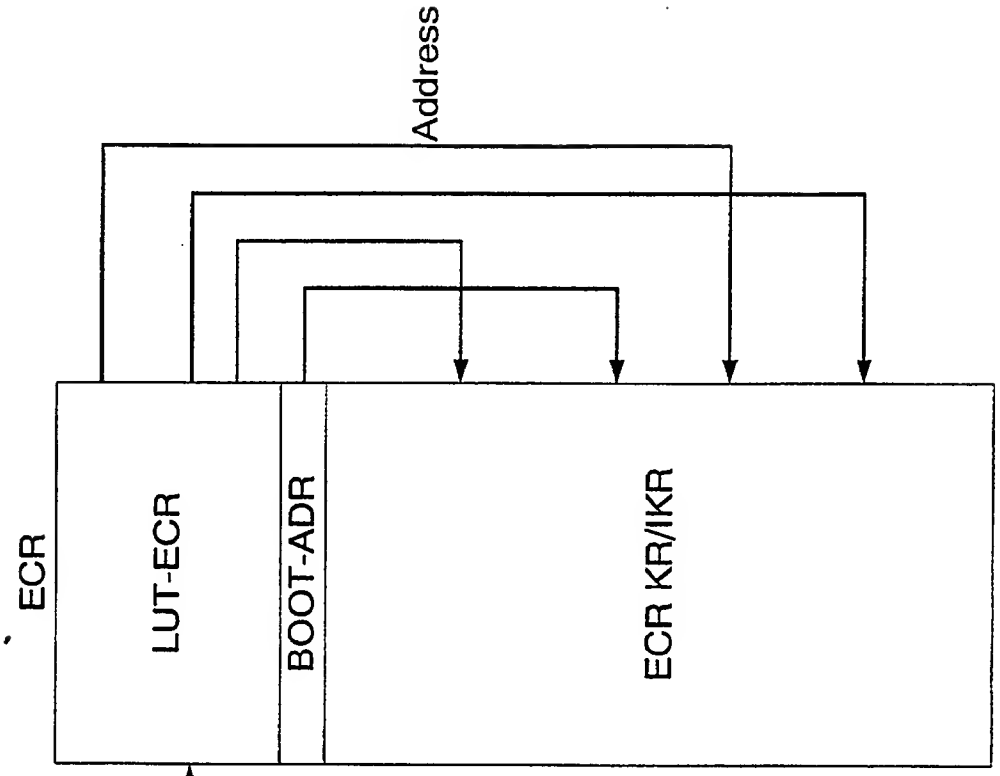


Fig. 16b

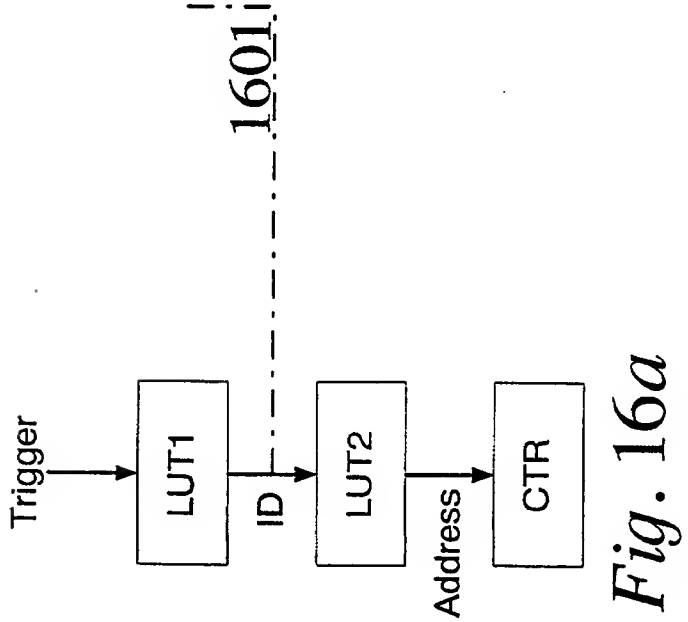


Fig. 16a

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

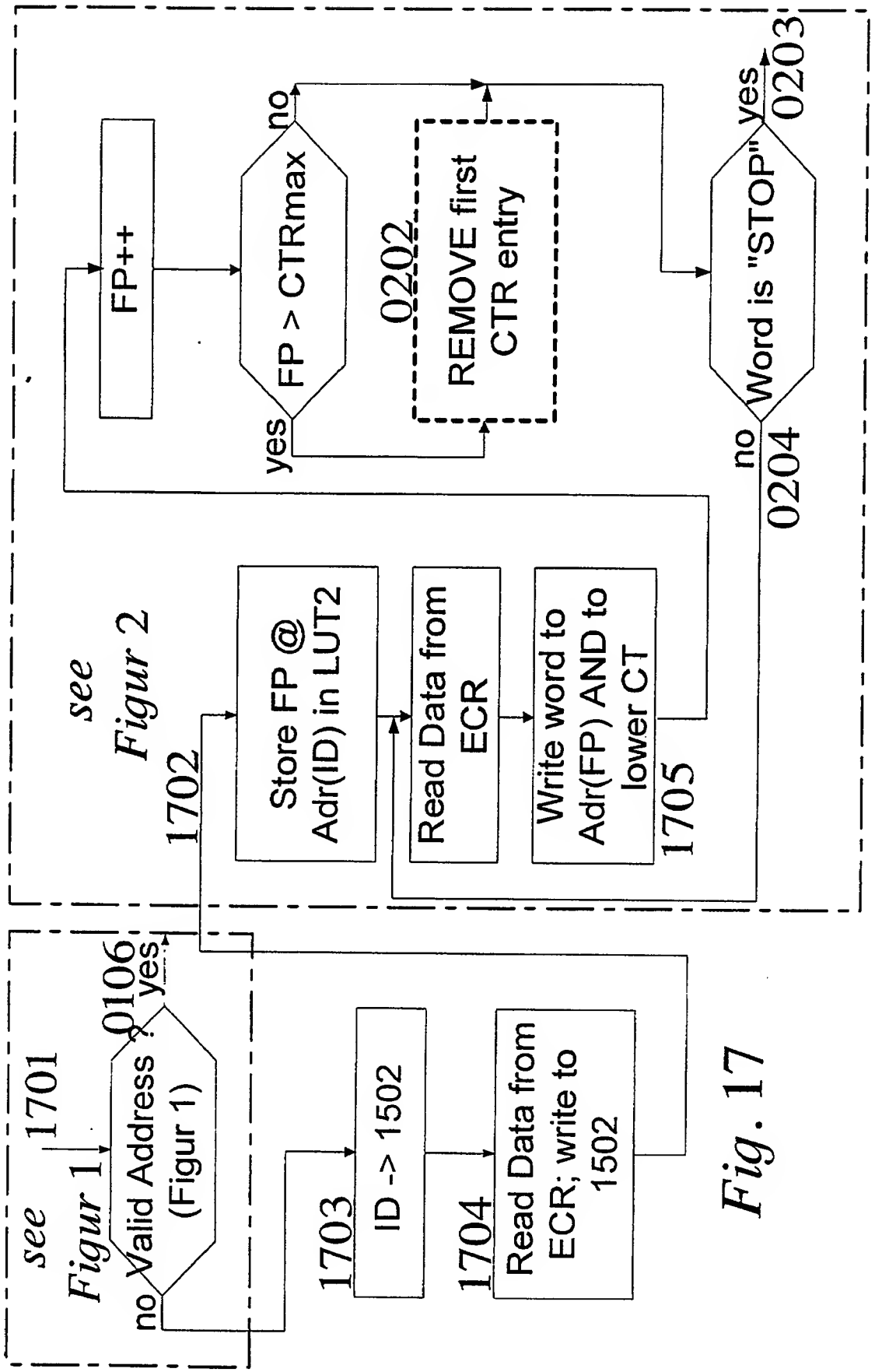


Fig. 17

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

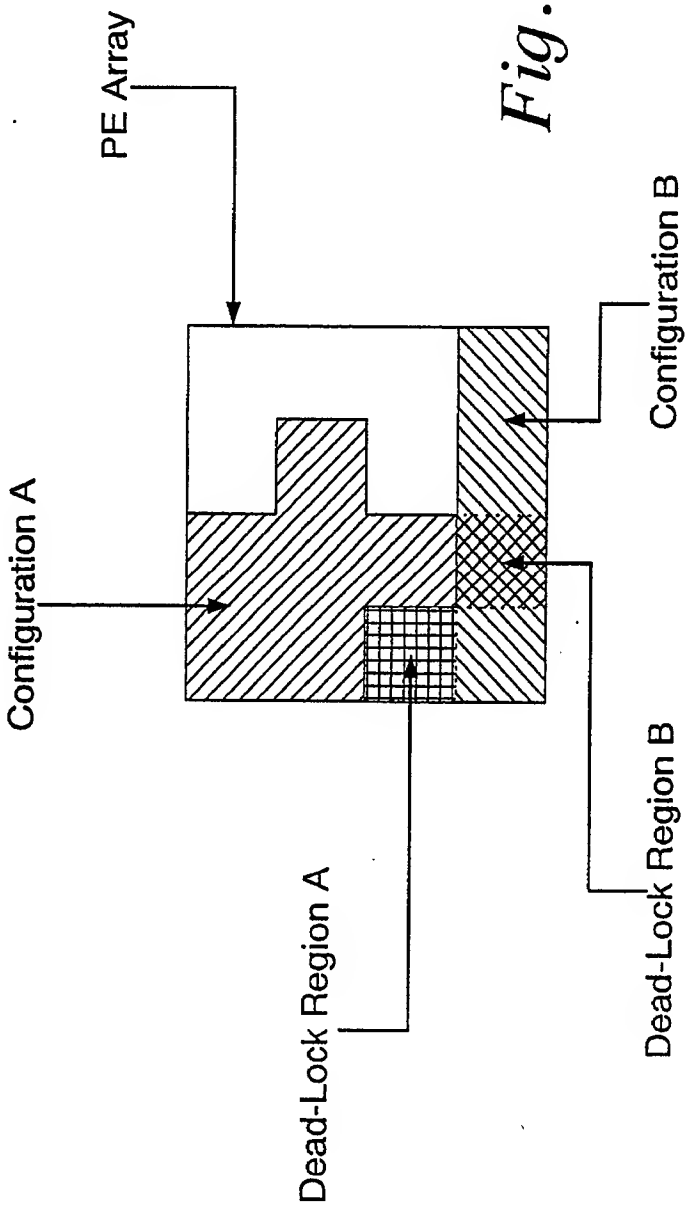


Fig. 18

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

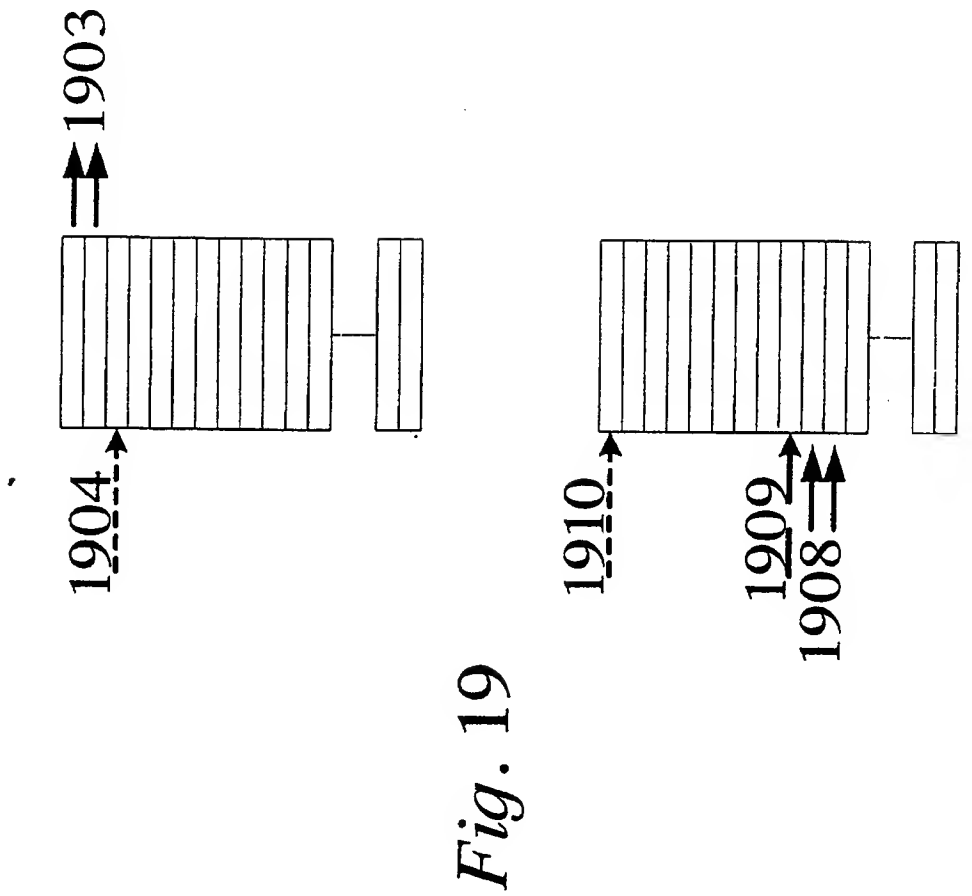


Fig. 19

Nummer: DE 198 07 872 A1
Int. Cl.[6]: G 06 F 15/80
Offenlegungstag: 26. August 1999

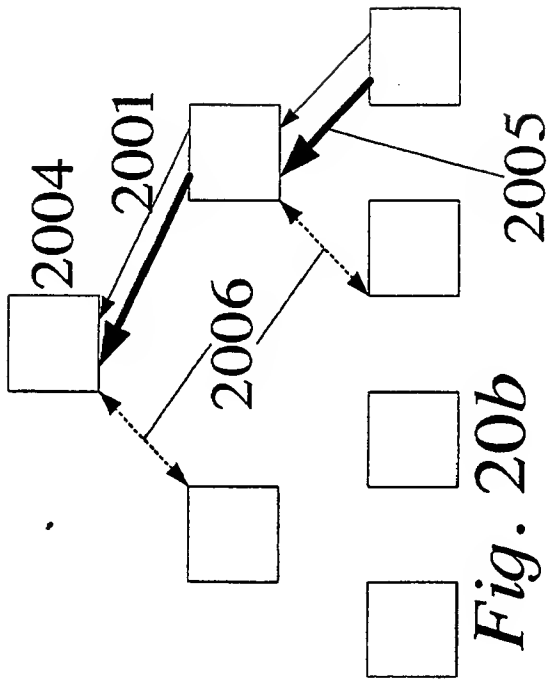


Fig. 20b

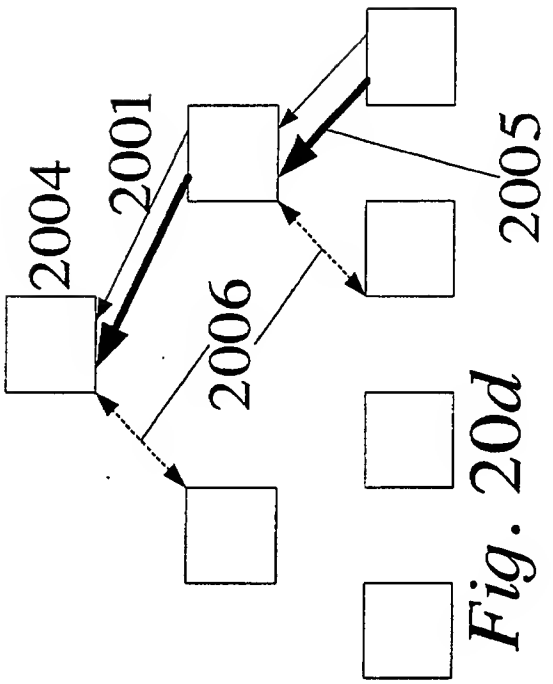


Fig. 20d

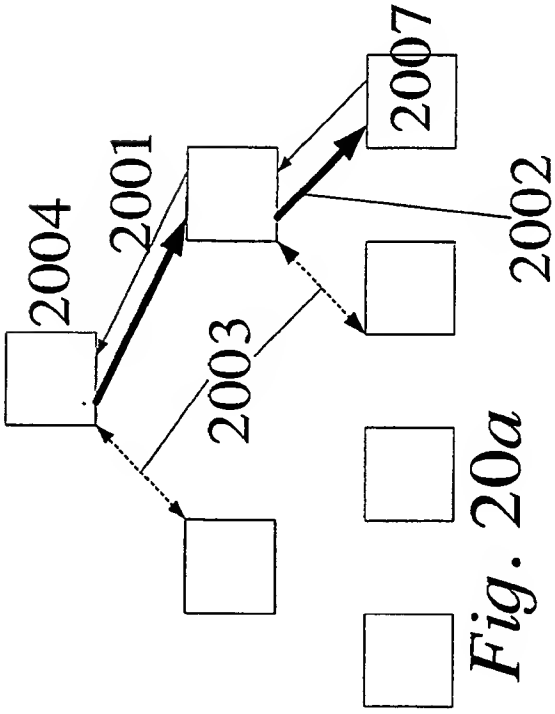


Fig. 20a

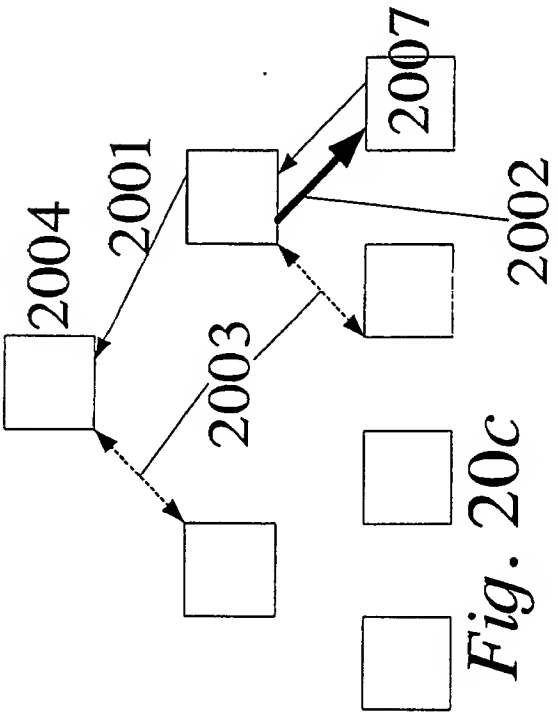


Fig. 20c